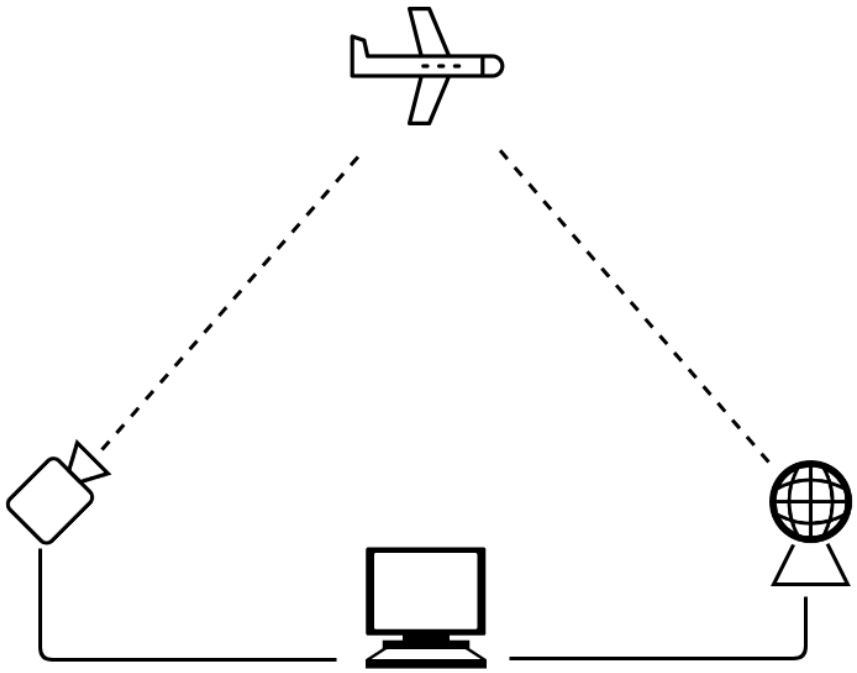
Masteroppgave

NTNU
Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

Peder Hogstad Birkeland

Kamerabasert Posisjoneringssystem for Fly og Helikopter

Masteroppgave i Kybernetikk og Robotikk
Veileder: Annette Stahl
Juni 2019



NTNU
Norwegian University of
Science and Technology

FFI Forsvarets forskningsinstitutt

Peder Hogstad Birkeland

Kamerabasert Posisjoneringssystem for Fly og Helikopter

Masteroppgave i Kybernetikk og Robotikk
Veileder: Annette Stahl
Juni 2019

Norges teknisk-naturvitenskapelige universitet
Fakultet for informasjonsteknologi og elektroteknikk
Institutt for teknisk kybernetikk

NTNU
Norwegian University of
Science and Technology

Til venner
Til familie
Til Sing
Til Rebekka

Oppsummering

Denne masteroppgaven tar for seg arbeidet med å utvikle og teste et kamerabasert sensorsystem for posisjonering av fly og helikoptre i samarbeid med Forsvarets Forskningsinstitutt (FFI).

Systemet som er utviklet består av to kamera som begge er koblet til en felles datamaskin for prosessering. Det ene kameraet er et 360°-kamera som kan overvåke himmelen i alle retninger samtidig. Det andre kameraet har et smalere synsfelt og er montert på et styrbart hode slik at det kan pekes i alle mulige retninger.

For å hente ut informasjon om flys posisjon fra bildene kameraene produserer kjøres forgrunnsdeteksjon på bildene fra 360°-kameraet. Denne informasjonen blir så brukt til å styre det andre kameraet mot målet. Informasjonen fra begge disse kameraene smeltes sammen til å triangulere objektets nøyaktige posisjon.

Oppgaven tar for seg de nøyaktige maskinvarekomponentene som ble valgt til å utgjøre systemet og hvorfor disse er hensiktsmessige til dette bruksområdet. Den inneholder også en grundig beskrivelse av de metodene som er tatt i bruk og hvordan disse er implementert i programvare som er blitt utviklet for dette systemet.

Til sist er systemet blitt testet og dets funksjon og nøyaktighet er analysert. På bakgrunn av denne testen konkluderes det med at systemet i sin nåværende form har potensiale, men har betydelige feil som må utbedres før systemet kan tas i bruk.

Abstract

This master thesis concerns the development and testing of a camera based sensor system for the purpose of positioning aircraft in cooperation with the Norwegian Defence Research Establishment (FFI).

The system consists of two cameras, connected to a shared computer for processing. One camera is a 360°-camera which can monitor the sky in every direction simultaneously. The other has a narrower field of view and is mounted to a controllable head such that it may be pointed in any direction.

Background subtraction is used in order to extract information about the position of aircraft in the images from the 360°-camera. This information is then used to point the other camera in the direction of the target. The information produced by both cameras are then fused in order to triangulate the exact position of the target.

The thesis covers the exact hardware components which were chosen for the system and why these are appropriate for this use. It also contains a thorough description of the methods the system utilizes and how these were implemented in the software which was developed for this platform.

Lastly the system has been tested and its prescision and function has been analyzed. Based on the results of this test it has been concluded that the system in its current form has potential. However, it also has considerable flaws which must be improved before the system can be truly useful.

Forord

Denne oppgaven konkluderer mitt 5-årige studium ved programmet Kybernetikk og Robotikk med hovedprofil Robotsystemet ved NTNU. Oppgaven er skrevet i samarbeid med Forsvarets Forskningsinstitutt (FFI) og bygger på min sommerjobb der i 2018 samt prosjektoppgaven som ble skrevet høsten 2018.

Under arbeidet med oppgaven har mine veiledere ved FFI, Jan-Kenneth Bekkeng, Harald Hovland og Alexander Wold hjulpet med utviklingen av systemet, faglige diskusjoner og tilbakemeldinger på denne oppgaven. Denne hjelpen har vært uvurderlig for mitt arbeid dere fortjener en stor takk. En takk rettes også til min veileder ved NTNU, Annette Stahl, som også har gitt tilbakemeldinger på oppgaven.

Det er FFI som har stilt alt av nødvendig program- og maskinvare til min disposisjon under arbeidet med denne oppgaven. Dette inkluderer begge kameraene som er brukt, datamaskinen som har fungert som sentralsystem og Visual Studio som har blitt brukt til å utvikle programvaren systemet kjører. FFI har også gitt meg en arbeidsplass i deres lokaler samt et sted å bo mens arbeidet har krevd min tilstedeværelse ved FFI. Dette er jeg svært takknemlig for. Det har vært fantastisk og lærerikt å få jobbe så tett med FFI over nesten et helt år og det har vært utrolig gøy å få jobbe praktisk med slikt dyrt og avansert utstyr.

Under sluttarbeidet med oppgaven ble det utført en test av systemet med en drone. Her vil jeg takke FFI for lån av nødvendig utstyr, som drone og DGPS. Denne testen hadde ikke vært gjennomførbar uten Jonas Moen og Tor Atle Solend som hjalp til med forsøket. De utførte det nødvendige oppsettet for DGPSen og styrte dronen under forsøket. Tusen takk til dere begge. Jeg er utrolig takknemlig for at dere tok dere tid til å hjelpe med dette forsøket.

Peder Hogstad Birkeland

Innhold

Figurer

Forkortelser

10GigE	=	10 Gbit Ethernet
API	=	Application Programming Interface
DGPS	=	Differential Global Positioning System
ENU	=	East North Up
GPIO	=	General Purpose Input Output
GPS	=	Global Positioning System
LTI	=	Linear Time Invariant
MOG	=	Mixture Of Gaussians
PID	=	Proportial Integral Derivative
PTU	=	Pan Tilt Unit
SDK	=	Software Development Kit
USB	=	Universal Serial Bus

Kapittel 1

Introduksjon

I dette kapittelet vil denne oppgaven introduseres. Det er tanken at kapittelet skal gi leseren forståelse av bakgrunnen for oppgaven, hvilket problem det forsøkes å løses, og hva som er oppgavens målsetning. Det vil også bli gitt en kort forklaring av oppgavens bidrag og struktur.

1.1 Bakgrunn

Sommeren 2018 tok jeg en stilling som forskningsassistent ved Forsvarets Forskningsinstitutt (FFI) som en del av den nødvendige praksisen som kreves for mitt masterstudium innen kybernetikk og robotikk. Jeg ble satt til å arbeide under Harald Hovland, Jan-Kenneth Bekkeng og Alexander Wold på et prosjekt som omhandlet bruken av et 360°-kamera FFI hadde gått til anskaffelse av. Min oppgave ble å analysere bildene fra kameraet for å finne flyvende objekter og følge disse objektenes bane gjennom bildene. Dette kunne så brukes, sammen med en antagelse om kjent flyvehøyde til å estimere posisjonen til objektene. Dette arbeidet ble utført over en periode på 8 uker og ble avsluttet med en felttest av systemet.

Jeg har fått være så heldig å få fortsette å jobbe med dette prosjektet videre, også etter at jobben som forskningsassistent hos FFI var over. Arbeidet fortsatte i første omgang som min prosjektoppgave i høstsemesteret 2018, nå med Annette Stahl fra Institutt for Teknisk Kybernetikk med som veileder fra NTNU. Oppgaven her var å beskrive systemets virkemåte, analysere dets presisjon og undersøke mulige forbedringer. Oppgaven tilsvarte 15 studiepoeng og ble avsluttet i desember 2018. Denne oppgaven belyste flere mangler ved systemet og pekte særlig ut antagelsen om kjent flyvehøyde som en stor begrensning.

Da oppgaven skulle videreføres som min masteroppgave i vårsemesteret 2018 var derfor hovedfokus på å fri systemet fra denne antagelsen. Valget falt på å gjøre dette ved å inkludere nok et kamera i systemet for å tillate triangulering av objektsposisjonen. Dette arbeidet, sammen med denne rapporten, tilsvarer 30 studiepoeng og ble avsluttet i juni 2019.

Denne rapporten er konklusjonen av dette prosjektet for min del. Den beskriver i detalj den nåværende oppbyggingen og sammensetningen av systemet og bedømmer dets presisjon og brukbarhet.

1.2 Problemstilling

Dette systemet er i utgangspunktet tenkt brukt i sammenheng med FFIs tester av motmidler mot varmesøkende missiler beregnet på fly eller helikoptre. I slike tester velges et punkt i luften hvor motmiddelet skal iverksettes og aparater og sensorer som skal gjøre målinger av motmiddelet rettes mot dette punktet. Et fly, eller helikopter, skal så fly gjennom dette punktet og iverksette motmiddelet idet det passerer gjennom det valgte punktet. Om flyet bommer på punktet eller iverksetter motmiddelet for sent eller tidlig vil ikke målingene som gjøres være fullstendig nøyaktige, da de bruker antagelsen om at motmiddelet ble iverksatt på det gitte punktet.

Hensiktet med systemet som denne oppgaven omhandler er å måle flyets posisjon, med høy nøyaktighet. Disse målingene skal helst produseres i sanntid, men det er også ønskelig at dataen fra systemet skal kunne lagres slik at opptaket kan analyseres i ettertid. Det er også viktig at målingene gjøres med en frekvens på 10 Hz eller mer. For å produsere disse målingene skal det tas i bruk to kameraer, ett som ser i alle retninger samtidig, og ett som kan styres til å se i alle retninger. Med kunnskap om egenskapene til disse kameraene skal opptak fra disse være tilstrekkelig til å beregne posisjonen til de objektene som er synlig for begge kameraene.

Målingene som systemet gjør kan da brukes både til å korrigere flyet på dets vei mot det fastsatte punktet, samt å finne den faktiske posisjonen til punktet hvor motmiddelet ble iverksatt. Dette krever en mest mulig automatisert kalibrering som del av programvaren systemet kjører.

1.3 Mål for Oppgaven

Målet for denne oppgaven er å utvikle et system som kan produsere nøyaktige posisjonsmålinger av fly og helikoptre i sanntid. Det er ønskelig at posisjoneringsfeilen skal ligge under 2 m for objekter på en avstand opp til 2 km fra systemet. På avstander lengre enn dette kan en høyere feil tolereres, men objekter skal kunne følges og posisjoneres helt opp til 5 km avstand. Det er ønskelig at disse målingene skal kunne gjøres med høy frekvens og i omtrent sanntid.

Det er også ønskelig at dette systemet skal være enkelt å ta i bruk og kunne fungere i generelle miljø. Videre er det en fordel om systemet i stor grad er automatisert og fungerer autonomt, da dette reduserer personellbehov. Målet er altså at systemet skal ta kort tid å sette opp og deretter ikke trenge menneskelig innblanding. Det skal heller ikke være nødvendig med omfattende kalibrering eller justering av systemet som krever detaljkunnskap om systemets virkemåte.

1.4 Bidrag

Denne oppgaven beskriver i detalj de metodene som er brukt og utviklet for å oppnå et system som kan posisjonere flyvende objekter ved hjelp av to kamera. Det beskrives hvilke maskinvarekomponenter som er tatt i bruk og hvordan informasjonen fra disse er brukt til å beregne objektenes posisjon. Det innovative ligger i hovedsak i hvordan et 360°-kamera og et styrbart kamera med smalere synsfelt er satt sammen i et system. Dette inkluderer også bruken av forgrunnsdeteksjon til å lokalisere potensielle mål.

Videre følger en beskrivelse av hvordan funksjonaliteten og nøyaktigheten til systemet er blitt testet og en diskusjon av hva disse testene viste. Til sist diskuteres det hvilke utfordringer systemet har og hvordan disse muligens kan forbedres i videre arbeid.

1.5 Struktur

Resten av denne oppgaven er strukturert på følgende måte. Først kommer et kapittel som beskriver noen grunnleggende metoder og algoritmer som er essensielle for systemets funksjon. Hvordan disse er benyttet og hva som oppnås med dem er beskrevet i kapittel 3, som tar for seg hvordan systemet er bygd opp. Dette kapittelet inneholder fire hoveddeler.

Den første delen er en beskrivelse av hvordan systemet er blitt simulert på forhånd for å prioritere de feilkildene med størst utslag på resultatene. De to neste delene beskriver arbeidet som er blitt gjort for å sette sammen systemet. Disse er delt inn i maskinvare og programvare. Maskinvare-kapittelet tar for seg hvilke fysiske komponenter som er blitt valgt til å utgjøre systemet og hvorfor. Programvare-kapittelet beskriver den programvaren systemet kjører. Gjennom dette belyses hvordan systemet bruker de forskjellige fysiske komponentene og hvordan dataen fra dem behandles. Dette viser også den generelle informasjonsflyten i systemet og alle stegene som tas for å komme frem til det endelige posisjonsestimatet.

Etter dette kommer et kapittel som beskriver hvordan systemet har blitt testet og legger frem resultatene av disse testene. Disse resultatene og deres betydning blir så diskutert i kapittel 5. Til sist følger en konklusjon basert på de utførte testene og deres resultater før det diskuteres hvilke forbedringer eller utvidelser det er mulig å inkludere i systemet.

Kapittel 2

Bakgrunnsteori

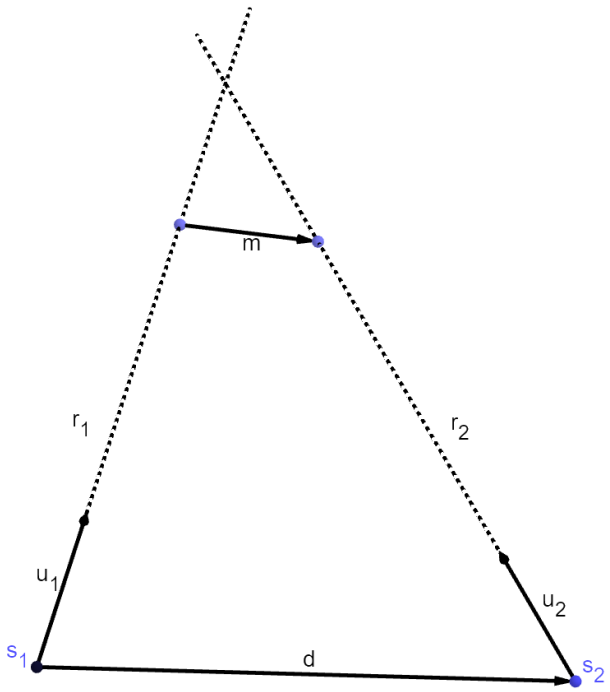
Denne oppgaven tar for seg mange tekniske detaljer om hvordan de ulike systemene i dette prosjektet er blitt implementert. Selv som det antas at leseren har en viss kjennskap til programmering, bildebehandling, og matematikk skal det ikke være nødvendig med detaljkunnskap for å forstå oppgavens innhold. Av denne grunn vil det i dette kapittelet legges frem et fåtall algoritmer og metoder som systemet baserer sin virkemåte på. Forståelsen av disse vil være nødvendig for full forståelse av oppgavens videre innhold.

2.1 Triangulering

Som nevnt innledningsvis vil denne oppgaven basere seg på informasjon fra to kamera rettet mot et felles mål. Informasjonen fra disse kameraene vil være to linjer i rommet som peker fra hvert kamera mot dette felles målet. Systemet må da være istand til å tolke disse to linjene og estimere målets posisjon ut i fra dette. En algoritme for dette formålet er trianguleringsalgoritmen som legges frem her. Innholdet i dette delkapittelet er basert på [2].

Først skal dette systemet defineres matematisk. Det er to kameraposisjoner, kalt $\mathbf{s_1}$ og $\mathbf{s_2}$. For enkelhetsskyld defineres $\mathbf{s_1}$ til å være origo i koordinatsystemet og det antas at begge kameraene opererer med koordinatsystem som ikke er rotert i forhold til hverandre. Definer da $\mathbf{d}$ som en vektor som peker fra $\mathbf{s_1}$ til $\mathbf{s_2}$ og gir transformasjonen mellom de de to kameraenes lokale koordinatsystem.

Videre vil hvert kamera ha en vektor som har utspring i hver kameraposisjon og som peker mot målet, anta for videre utregnings enkelhet at disse er enhetsvektorer, kalt $\mathbf{u_1}$ og $\mathbf{u_2}$. Se figur 2.1. Husk at dette er et tredimensjonalt system, illustrert med en todimensjonal figur. Det at linjene krysser i figuren betyr ikke at de gjør det i tre dimensjoner og punktet hvor de krysser i figuren er heller ikke nødvendigvis der de to linjene kommer nermest hverandre i rommet.



Figur 2.1: Oppsett for triangulering

Da det må tas høyde for støy i systemet kan det antas at $\mathbf{u_1}$ og $\mathbf{u_2}$ ikke ligger i et felles plan og dermed at linjene de definerer ikke møtes i et felles punkt. Målet blir derfor ikke å finne hvor disse linjene krysser, da det antas at de ikke gjør det, men heller å finne de to punktene på linjene som er nærmest hverandre. Dette kan defineres matematisk. De to punktene som skal finnes kan beskrives som $r_1\mathbf{u}_1$ og $r_2\mathbf{u}_2$ hvor r_n definerer avstanden mellom kamera n og det punktet som skal finnes langs linjen fra dette kameraet. Kall vektoren mellom disse punktene $\mathbf{m}$, definert

$$\mathbf{m} = r_1\mathbf{u_1} - \mathbf{d} - r_2\mathbf{u_2}. \tag{2.1}$$

Målet blir altså å velge r_1 og r_2 slik at $|\mathbf{m}|$ blir minst mulig. Dette kan gjøres med minste kvadraters metode ved å minimere for $\mathbf{m} \cdot \mathbf{m}$. Det resulterende problemet kan beskrives som

$$\mathbf{Hx} - \mathbf{d} = \mathbf{m}, \tag{2.2}$$

hvor $\mathbf{m}$ er vektoren som skal minimeres, og

$$\mathbf{H} = \begin{bmatrix} \mathbf{u_1} - \mathbf{u_2} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} r_1 \\ r_2 \end{bmatrix}.$$

Løsningen på dette minste kvadrat-problemet er gitt ved

$$\hat{\mathbf{x}} = \begin{bmatrix} \hat{r}_1 \\ \hat{r}_2 \end{bmatrix} = (\mathbf{H}^\mathsf{T}\mathbf{H})^{-1}\mathbf{H}^\mathsf{T}\mathbf{d}, \tag{2.3}$$

der $\hat{r}_1$ og $\hat{r}_2$ er de avstandene fra hvert kamera som minimerer lengden av $\mathbf{m}$. Med antagelsen om at hver linje er like nøyaktig vil det beste estimatet av objektets faktiske posisjon være punktet midt mellom de punktene, gitt ved

$$\hat{\mathbf{o}} = \frac{\hat{r}_1\mathbf{u_1} + \hat{r}_2\mathbf{u_2}}{2}, \tag{2.4}$$

hvor $\hat{\mathbf{o}}$ er estimatet av objektets posisjon.

Med dette er det produsert et godt estimat av objektets posisjon. Dette estimatet er kun basert på kameraenes relative posisjon, og informasjon som kan finnes ved å analysere bildene fra kameraene.

Som nevnt må det medregnes at de to linjene som peker mot målet ikke er helt nøyaktige. Derfor vil heller ikke det produserte estimatet av objektets posisjon være helt nøyaktig. Det er ønskelig å vite noe om hvor stor denne unøyaktigheten er og hvilke faktorer den påvirkes av. Definer ψ til å være vinkelen mellom de to retningsvektorene i planet de spenner. Da er, ifølge [2], variansen for avstandsestimatet $\hat{r}_1$ gitt ved

$$\sigma_{r_1}^2 = \frac{\cos^2(\psi) r_1^2 \sigma_{\theta_1}^2 + r_2^2 \sigma_{\theta_2}^2}{\sin^2(\psi)}. \tag{2.5}$$

Og tilsvarende for $\sigma_{r_2}^2$. Hvor $\sigma_{r_1}^2$ og $\sigma_{r_2}^2$ er variansen i henholdsvis r_1 og r_2, og $\sigma_{\theta_1}^2$ og $\sigma_{\theta_2}^2$ er variansen til feilen i de to retningsvektorene.

Denne algoritmen danner et viktig grunnlag for funksjonen til systemet denne oppgaven omhandler. Den vil være nødvendig for å kunne estimere objekters posisjon samt si noe om nøyaktigheten av disse estimatene.

2.2 Forgrunnsdeteksjon

Det ene av kameraene som dette prosjektet tar i bruk er et 360° kamera. Dette kameraet kan observere hele himmelen mens det står fullstendig i ro. Systemet må kunne detektere mulige mål på himmelen og skille disse fra bakgrunnen i bildene. Dette kan gjøres på flere måter, for eksempel enkel subtraksjon. Siden systemet må kunne operere utendørs er det nødvendig å ta hensyn til at ting i bakgrunnen, som trær eller skyer, vil kunne ha litt bevegelse uten at de er potensielle mål. For å håndtere dette trengs en mer sofistikert algoritme for forgrunnsdeteksjon. Det vil være tema i dette delkapittelet som baserer seg på artiklene [3] og [7].

For å belyse problemet med forgrunnsdeteksjon, og hva en god forgrunnsdeteksjonsalgoritme må ta høyde for, vil det her presenteres et eksempel. Si at det foreligger en bildestrøm fra et stasjonært kamera, som filmer en stasjonær scene. På et tidspunkt passerer et objekt, for eksempel en person, gjennom scenen slik at det plukkes opp av kameraet. Målet med en forgrunnsdeteksjonsalgoritme er da å produsere en kopi av denne bildestrømmen hvor alle piksler er markert enten som forgrunn eller bakgrunn. Objektet som beveger seg gjennom scenen er et eksempel på et forgrunnsobjekt, resten vil være bakgrunn.

Siden kameraet står stille vil alle bilder av kun scenen se veldig like ut. For å skille ut de områdene hvor noe har beveget seg gjennom scenen er det nok å sammenligne et bilde fra kameraet med ett som er tatt når det er bekreftet at det ikke er noen forgrunnsobjekter til stede. Det bildet som ikke inneholder noen forgrunnsobjekter kalles da bakgrunnsmodellen. Når det skal analyseres om et forgrunnsobjekt er tilstede i et bilde analyseres differansen mellom hver piksel i dette bildet med tilsvarende piksel i bakgrunnsmodellen. Der hvor forskjellen er stor kan det konkluderes med at opphavet til pikselet er et forgrunnsobjekt og ikke bakgrunnen.

Dette er en veldig enkel form for forgrunnsdeteksjon og den har dermed også store mangler. Den største av disse er at metoden ikke er robust mot endringer i lysforhold. Om scenen som betraktes har noen form for naturlig lys vil dette forandre seg over tid. Dette fører til store endringer av hele bildet som vil gi utslag som forgrunn selv om det kun er bakgrunnen som forandres. En måte å kompensere for dette er å alltid bruke bilder som er relativt nylige som bakgrunnsmodell, men også dette har flere ulemper. Ved å hele tiden bytte det bildet som utgjør bakgrunnsmodellen blir det vanskeligere å garantere at det ikke er objekter tilstede. Dette er heller ikke robust mot mer plutselige endringer i lys, som at solen går bak en sky.

For å kompensere for dette er det mulig å heller bruke en adaptiv bakgrunnsmodell å sammenligne med, istedet for et vanlig stillbilde. Dette betyr at algoritmen bruker n av de siste bildene som kameraet har tatt og bruker dette til å si noe om hvordan bakgrunnen ser ut. Når et nytt bilde blir tatt forkastes det eldste av de n bildene og erstattes med det nyeste. Hvis et nytt bilde kommer inn med et område som er svært forskjellig fra bakgrunnsmodellen kan det konkluderes at dette er et forgrunnsobjekt og at denne delen av bildet ikke skal brukes til å oppdatere bakgrunnsmodellen. Dette gjør at algoritmen vil tilpasse seg endringer i lysforhold og fortsatt kunne oppdage forgrunnsobjekter fort.

Et annet problem som denne algoritmen ikke tar høyde for er spørsmålet: hvor forskjellig er forskjellig nok? Selv to bilder som tas rett etterhverandre av et stasjonært kamera vil ha små forskjeller mellom seg grunnet kamerastøy. Dette gjør at hele bildet hele tiden vil ha små forandringer fra bakgrunnsmodellen. Hvordan kan algoritmen da vite om forskjellen den registrerer skyldes støy eller et forgrunnsobjekt? Denne terskelen må settes på en måte og det beste vil være om algoritmen kan finne den av seg selv. Hvor mye støy et kamera har avhenger nemlig både av kvaliteten på kamerasensoren og hvor godt scenen er belyst. En god måte å gjøre dette på er å bektrakte hvert piksel i bakgrunnen som en normalfordelt stokastisk variabel med forventningensverdi μ og standardavvik σ. Da vil hvert bilde av bakgrunnen kunne betraktes som en måling av denne variabelen og estimatorer for de faktiske verdiene kan defineres ved

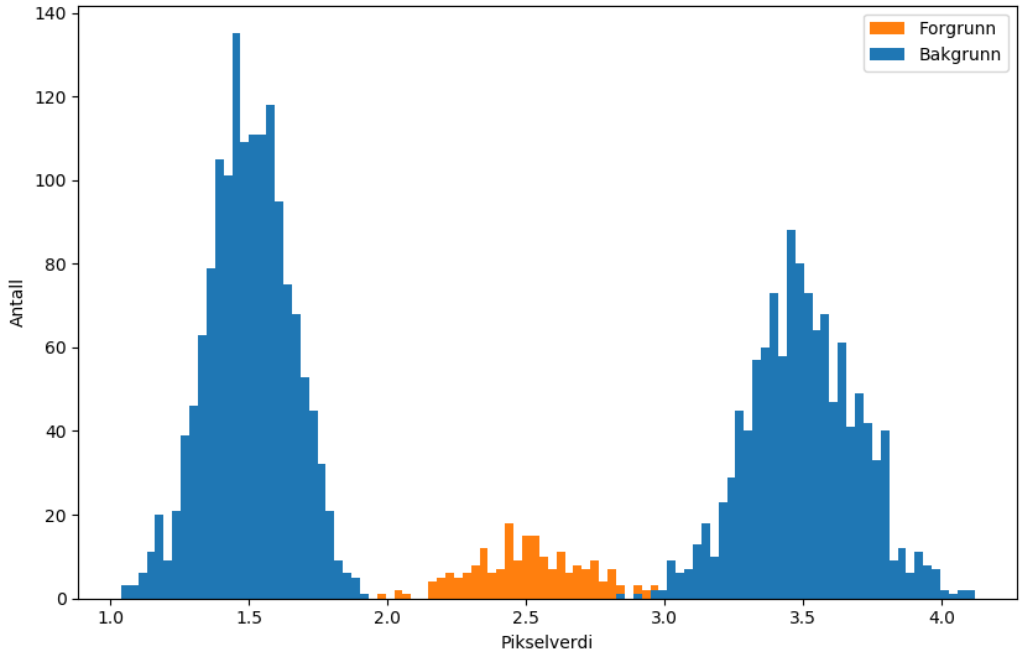
$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} p_i, \quad \hat{\sigma} = \sqrt{\frac{\sum_{i=1}^{n}(\hat{\mu} - p_i)^2}{n-1}}. \tag{2.6}$$

Der $\hat{\mu}$ er estimatet av μ, $\hat{\sigma}$ er estimatet av σ, p_i er observert pikselverdi i bilde i, og n er antall bilder som brukes til å estimere bakgrunnen. Når et nytt bilde sendes inn til algoritmen kan den enkelt sjekke om den observerte pikselverdien er oppstått utifra denne fordelingen eller ikke. Med andre ord, hvis det holder at $|\hat{\mu} - p| \gg \hat{\sigma}$, er det svært sannsynlig at opphavet til dette pikselet er er forgrunnsobjekt, ikke bakgrunnen.

Algoritmen begynner nå å bli mer mer kompleks, men det er fortsatt ting den ikke tar høyde for, og det er kontinuerlig variasjon av bakgrunnen. Anta at scenen som betraktes er utendørs og inneholder trær, gress og kanskje litt vann. Når vinden blåser vil disse tingene bevege på seg. Ta eksempelet med et blad på et tre. Dette bladet vil bevege på seg slik at noen piksler rundt bladets kant vil være enten det grønne bladet, eller den blå himmelen bak. Dette betyr at dette pikselet har en stor variasjon i hvilke verdier det vil ha fra bilde til bilde, selv om det ikke er noen forgrunnsobjekter tilstede. Det er ønskelig at algoritmen skal markere dette pikselet som bakgrunn, uavhengig av om det observerer bladet eller himmelen bak. Med den bakgrunnsmodellen som er lagt frem til nå vil dette kreve at $\hat{\sigma}$ velges vært høy. Dette har ulempen av hvis et forgrunnsobjekt faktisk beveger seg foran dette pikslet vil ikke $|\hat{\mu} - p| \gg \hat{\sigma}$ være oppfylt da $\hat{\sigma}$ er så stor.

Løsningen på dette problemet er å bruke en sammensetning av normalfordelinger (eng. mixture of gaussians (MOG)) til å representere hvert piksel. Da kan, i dette eksempelet, pikselet ha to normalfordelinger, en som beskriver bladet og en som beskriver himmelen bak. Om en pikselverdi observeres som ikke stemmer med noen av disse fordelingen kan det konkluderes med at pikslet representerer et forgrunnsobjekt. Dette har den fordelen at $\hat{\sigma}$ tilhørende hver normalfordeling kan holdes liten, slik at forgrunnsobjekter fortsatt lett kan oppdages selv om bakgrunnen er litt dynamisk. Moderne varianter av slike MOG forgrunnsdeteksjonsalgoritmer kan også adaptivt bestemme hvor mange normalfordelinger hvert piksel trenger for å beskrive bakgrunnen godt. Algoritmer for dette er beskrevet i [9] og [8]. Dette gjør at de pikslene som trenger det kan få mange fordelinger, uten å kaste bort ressurser ved å ha mange fordelinger der kun én er nødvendig.

I figur 2.2 vises et eksempel på et mulig histogram over en observert pikselverdi fra en film med en slik varierende bakgrunn som tidvis også inneholder et forgrunnsobjekt. Merk at selv om dette eksempelet er gitt i to dimensjoner vil som oftest denne algoritmen brukes på fargebilder. Da er ikke det mulige utfallsrommet for en piksel begrenset til kun en dimensjon, men vil strekke seg over tre dimensjoner.



Figur 2.2: Eksempel på pikselfordeling hvor mer enn en normalfordeling er nødvendig for å klassifisere bakgrunnen

Da systemet inneholder et kamera som er fullstendig stasjonært og har i oppgave å lokalisere objekter som beveger seg vil denne fremgangsmåten være svært nyttig for dette prosjektet. De mer komplekse delene av algoritmen, som diskutert mot slutten av dette delkapittelet, vil også ha stor betydning her da systemet er beregnet for utendørs bruk hvor vind og skyer er et uunngåelig faktum.

2.3 Kalman Filtrering

Dette systemet må ta høyde for at målinger og beregninger det gjør blir påvirket av støy. Det må også ta høyde for at det noen ganger ikke får inn målinger i det hele tatt. Ta eksempelet med et fly som følges over himmelen. Hvis flyet flyr bak en sky vil flyet en liten stund ikke synes i bildet, og ingen måling av dets posisjon vil kunne gjøres. Det er ønskelig at systemet skal kunne håndtere dette på en god måte. En algoritme som kan hjelpe både å redusere støy på målinger og gjøre gode prediksjoner når målinger ikke er tilgjengelig er et kalman filter. Dette delkapittelet vil legge frem hva et kalman filter er og hvordan det fungerer. I første omgang vil den generelle tanken bak kalman filteret diskuteres. Først mot slutten av dette delkapittelet vil de mer matematiske tilnærmingene til tema tas opp.

Kalman filteret er en algoritme som nå er blitt over 50 år gammel. Siden dens opprinnelse har metoden funnet svært mye bruk i alle mulige ingeniøremner og er høyt elsket av mange. Et av algoritmens bruksområder er i målfølging innenfor maskinsyn som er slik den er brukt i dette prosjektet. For å belyse det problemet kalman filteret forsøker å løse, ta dette eksempelet. Posisjonen til en bil som kjører langs en rett linje skal måles. Til å begynne med måles bilens posisjon, men ikke nøyaktig. Det antas at denne målingen er en forventningsrett estimator for bilens faktiske posisjon i målingsøyeblikket og at den er normalfordelt med et visst standardavvik. Med andre ord er

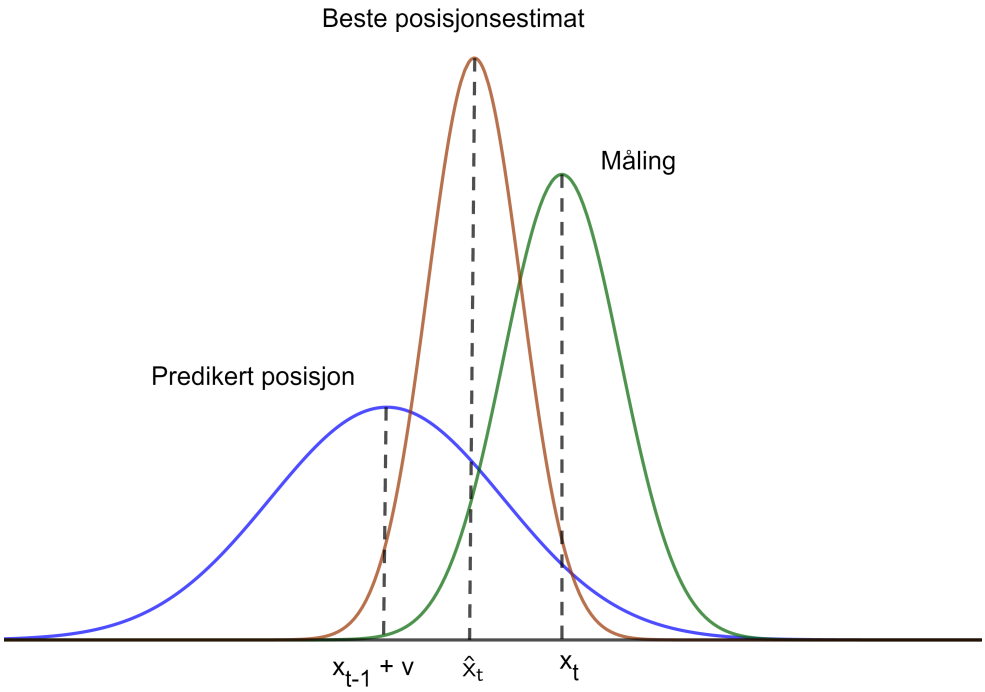
$$p(\hat{x}_0) \sim N(x_0, \sigma), \tag{2.7}$$

der $\hat{x}_0$ er målingen av bilens posisjon, x_0 er bilens faktiske posisjon, og σ er standardavviket til målingen. Anta nå at det finnes en modell for hvor fort bilen kjører, men at denne heller ikke er nøyaktig. Bilen vil noen ganger kjøre litt raskere eller litt saktere enn denne farten. Det kan antas at hvor langt bilen flytter seg på t tidsenheter også er normalfordelt, det vil si at

$$p(x_{t+1} - x_t) \sim N(v, \sigma_v), \tag{2.8}$$

der v er forventningsverdien til bilens hastighet og σ_v er dens standardavvik.

Anta videre at det etter t tidsenheter gjøres en ny måling av bilens posisjon, x_1, som har samme fordeling som beskrives i ligning 2.7. Det foreligger nå to målinger av bilens posisjon, en på tidspunkt 0 og en på tidspunkt 1. Det forligger også en modell for avstanden bilen av tilbakelagt mellom disse to tidspunktene. På et vis forligger det altså to estimater av bilens posisjon ved tidspunkt 1, både $\hat{x}_1$ og $\hat{x}_0 + v$. Hvor $\hat{x}_1$ har standardavvik σ og $\hat{x}_0 + v$ har standardavvik $\sigma + \sigma_v$. Det er da mulig og kombinere disse to estimatene for å oppnå et estimat med mindre standardavvik enn hver av de to målingene individuelt. Se figur 2.3. Når denne prosessen gjentas flere ganger over tid vil det også være mulig å raffinere målingene som gjøres. Med flere slike målinger vil det være mulig å gjøre bedre estimater av fordelingene som gir opphav, både til målingene og modellen til systemet.



Figur 2.3: Hvordan predikert estimat og måling kan kombineres til et bedre estimat

Dette er altså tanken bak et kalman filter og illustrerer den generelle virkemåten til algoritmen. Likevel er det vanskelig å implementere denne ideen uten en mer matematisk formulering. Dette vil presenteres nå og er basert på [1].

Kalmanfilteret baserer seg på den generelle formen for et diskret, lineært, system med både prosess- og målestøy. Beskrevet av

$$\mathbf{x}_{k+1} = \mathbf{A}_k\mathbf{x}_k + \mathbf{B}_k\mathbf{u}_k + \mathbf{w}_k \tag{2.9}$$

og

$$\mathbf{z}_k = \mathbf{H}_k\mathbf{x}_k + \mathbf{v}_k. \tag{2.10}$$

I denne notasjonen er $\mathbf{x}_k$ vektoren som inneholder alle systemets tilstander ved tidspunktet k. Disse tilstandene er ukjent og observeres kun gjennom målingsvektoren $\mathbf{z}_k$. Denne målingsvektoren er en transformasjon av $\mathbf{x}_k$ via matrisen $\mathbf{H}_k$ og påført støyvektoren $\mathbf{v}_k$.

$\mathbf{u}_k$ er vektoren som inneholder de ytre pådragene som påføres systemet ved tidspunkt k. Tilstandsvektoren ved neste tidssteg, $\mathbf{x}_{k+1}$, er da gitt av tre ledd. Det første er hvordan den interne tilstanden påvirker systemet videre. Dette er gitt av $\mathbf{A}_k\mathbf{x}_k$ hvor $\mathbf{A}_k$ er matrisen som bestemmer hvordan tilstanden ved tidspunkt $k+1$ påvirkes av tilstanden slik den var ved tidspunkt k. Neste ledd er $\mathbf{B}_k\mathbf{u}_k$ hvor matrisen $\mathbf{B}_k$ definerer hvordan det ytre pådraget ved tidspunkt k påvirker de indre tilstandene ved tidspunkt $k+1$. Det siste leddet er et støy-ledd.

Det antas at de to støyleddene er uavhengige, hvite og har fordelingen

$$p(\mathbf{w}) \sim N(0, \mathbf{Q}), \quad p(\mathbf{v}) \sim N(0, \mathbf{R}),$$

for alle k. Hvor $\mathbf{Q}$ og $\mathbf{R}$ er kovariansematrisene for elementene i henholdsvis $\mathbf{w}$ og $\mathbf{v}$.

For dette prosjektets formål holder det å begrense definisjonen til å gjelde et LTI-system, altså at

$$\mathbf{A}_k = \mathbf{A}_{k+1} = ... = \mathbf{A}, \quad \mathbf{B}_k = \mathbf{B}_{k+1} = ... = \mathbf{B}, \quad \mathbf{H}_k = \mathbf{H}_{k+1} = ... = \mathbf{H}.$$

Samtidig velges det her å bruke et autonomt system hvor $\mathbf{u}$ settes konstant lik 0. Da blir også $\mathbf{B}$ irrelevant og vil ikke tas hensyn til fremover. Dette tilsvarer et system som det ikke er mulig å påvirke.

Definer $\hat{\mathbf{x}}_k^-$ til å være estimatet av $\mathbf{x}$ på tidspunkt k, før målingen $\mathbf{z}_k$ tas i betraktning. Dette kalles a-priori estimatet og beregnes ved

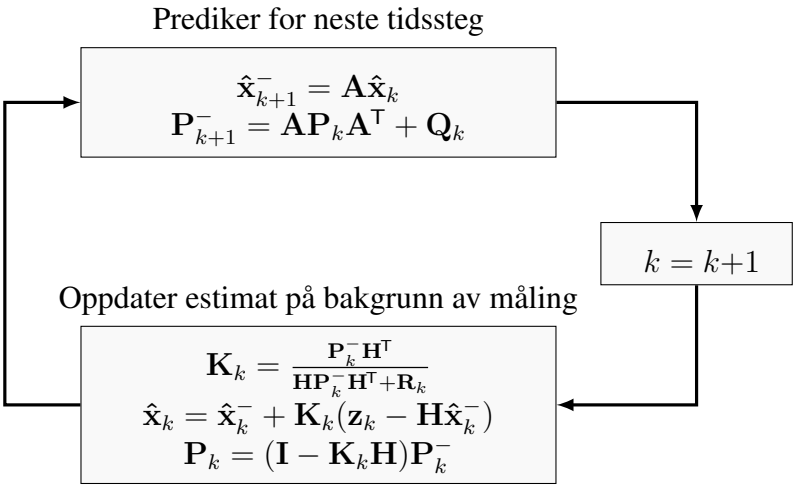
$$\hat{\mathbf{x}}_k^- = \mathbf{A}\hat{\mathbf{x}}_{k-1}, \tag{2.11}$$

hvor $\hat{\mathbf{x}}_k$ er det som kalles a-posteriori estimatet. Dette er estimatet hvor $\mathbf{z}_k$ er tatt med i betraktningen og beregnes ved en vektet sum av målingen og a-priori estimatet, spesifikt

$$\hat{\mathbf{x}}_k = \hat{\mathbf{x}}_k^- + \mathbf{K}_k(\mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_k^-). \tag{2.12}$$

Sentral i denne formelen er matrisen $\mathbf{K}$. Dette er en skaleringsmatrise som bestemmer hvor stor tillit skal ilegges målingen kontra a-priori estimatet. Mer spesifikt velges $\mathbf{K}$ slik at den minimerer a-posteriori kovariansematrisen definert ved

$$\mathbf{P}_k = E[\mathbf{e}_k\mathbf{e}_k^\mathsf{T}] \quad \text{hvor} \quad \mathbf{e}_k = \mathbf{x}_k - \hat{\mathbf{x}}_k. \tag{2.13}$$



Figur 2.4: Oppdateringssteg for kalman filter

Fremgangsmåten for å løse dette minimaliseringsproblemet er ikke diskutert her. Om leseren ønsker en gjennomgang av denne refereres det til [1]. For implementasjonens del holder det å vite at en $\mathbf{K}$ som løser problemet er gitt ved

$$\mathbf{K}_k = \frac{\mathbf{P}_k^-\mathbf{H}^\mathsf{T}}{\mathbf{H}\mathbf{P}_k^-\mathbf{H}^\mathsf{T} + \mathbf{R}_k}. \tag{2.14}$$

Bruken av filteret skjer da i to steg. Først kommer prediksjonssteget hvor verdiene for $\hat{\mathbf{x}}^-$ og $\mathbf{P}^-$ predikeres for neste tidssteg. Dette skjer via ligningene 2.11 og

$$\mathbf{P}_{k+1}^- = \mathbf{A}\mathbf{P}_k\mathbf{A}^\mathsf{T} + \mathbf{Q}_k. \tag{2.15}$$

Neste steg er å oppdatere disse estimatene basert på målingen som nå er gjort. Dette skjer ved ligningene 2.12, 2.14, og

$$\mathbf{P}_k = (\mathbf{I} - \mathbf{K}_k\mathbf{H})\mathbf{P}_k^-. \tag{2.16}$$

Se figur 2.4.

Dette er alt som skal til for å implementere et kalman filter. Merk at de formlene som er gitt her er spesifikt for et autonomt, diskret LTI-system. Innledningsvis ble det også nevnt muligheten for at det ved noen tidssteg ikke gjøres målinger. Da kan steg to av filteret sløyfes. Dette gjør at det fortsatt er mulig å få et godt estimat på objektets posisjon selv etter at det ikke har vært observert på en stund.

Med dette burde leseren ha fått en god forståelse for disse tre algoritmene som danner grunnlaget for virkemåten til dette systemet. I neste kapittel, som beskriver systemets oppbygging, vil det vises hvordan disse metodene er tatt i bruk. Dette kapittelet vil dermed også belyse hvorfor kjennskap til disse metodene er nødvendig for full forståelse av det sammensatte systemet.

Kapittel 3

Metode

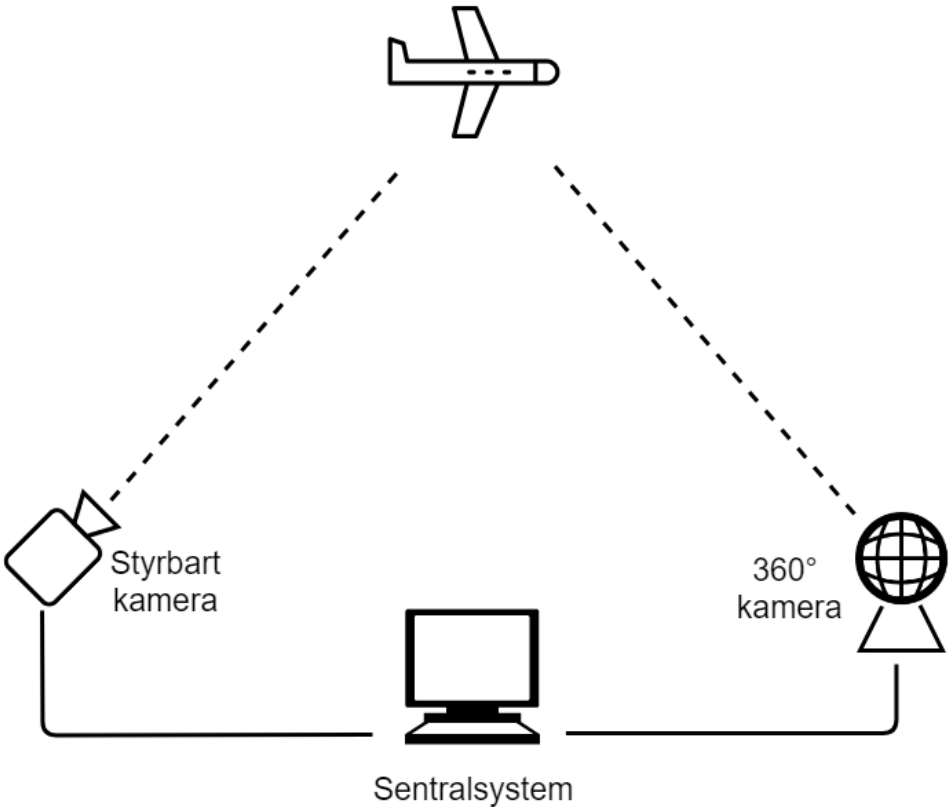
Målet for denne oppgaven er å sette sammen et system bestående av to kamera til å produsere målinger av posisjon til eventuelle flyvende objekter de to kameraene kan observere. Denne delen av oppgaven vil beskrive hvordan dette systemet produserer de målingene det gjør, hvilke deler det består av, og hvordan disse henger sammen.

Det første delkapittelet vil gi en generell beskrivelse av systemet og dets virkemåte, deretter følger et delkapittel dedikert til de maskinvarekomponentene som er tatt i bruk. Videre kommer et delkapittel om programvaren som er blitt utviklet for å oppnå systemets mål og hvordan denne er strukturert, og til sist kommer et kapittel om viktige aspekter av hvordan systemet skal settes opp og tas i bruk.

3.1 Overordnet Systembeskrivelse

Som nevnt innledningsvis er det ønskelig at dette systemet skal kunne detektere flyvende objekter på himmelen for så å produsere målinger som angir dette objektets posisjon. Objektene som skal posisjoneres vil i hovedsak være militære fly og helikopter. Det er ønskelig at disse objektene skal kunne detekteres og følges på minst 5 km avstand. Samtidig bør posisjoneringsfeilen holdes på 2 m eller mindre for avstander nermere enn 2 km.

Det er ønskelig at minst mulig antas kjent om objektet på forhånd. På bakgrunn av dette ønsket er det altså en fordel om så mye av himmelen som mulig kan overvåkes samtidig. Derfor vil det ene kameraet som inkluderes i systemet være et 360° kamera som kan se i alle retninger samtidig. Dette gjør at uansett hvor et fly dukker opp på himmelen vil det kunne bli oppdaget av dette kameraet.



Figur 3.1: Illustrasjon av sensorsystemet

Såfremt de interne egenskapene til kameraet, som fokallengden og sensorformatet, er kjent, vil det være mulig å kalkulere en romlinje som tilsvarer hver piksel i kameraets bilder. Med andre ord, om et piksel representerer et fly kan det beregnes en linje i rommet som tilsvarer dette pikselet. Flyet må da befinne seg på et punkt langs denne linjen.

Flyets mulige posisjoner er nå begrenset til en linje i rommet, men dette er åpenbart ikke nok til å fullstendig beskrive dets posisjon. Det er her systemets andre kamera kommer inn. Når det nå foreligger en viss begrensing i mulige posisjoner faller nødvendigheten med et 360°-kamera bort. Det er ikke nødvendig for dette kameraet å se i alle mulige retninger når det første kameraet tar seg av dette. Likevel trenger det andre kameraet muligheten til å se objektet, uavhengig av hvor det første kameraet oppdaget det. Dette kan løses ved å montere et vanlig kamera til et motorisert styrbart hode som kan peke i alle retninger.

Å velge et vanlig kamera fremfor et 360°-kamera som kamera nummer to har også andre fordeler. Et bilde fra et 360°-kamera dekker naturligvis et stort område. Når dette bildet har en gitt oppløsning betyr det at hvis det interessante området av bildet bare er en liten del, vil denne delen ha forholdsvis dårlig oppløsning. Ved å bruke et kamera med smalere synsfelt som kamera nummer to vil det også produseres mer høyoppløste bilder av målet.

La nå dette andre kameraet lete langs linjen produsert av det første kameraet. Når objektet oppdages av det andre kameraet kan det beregnes en ny linje mot målet. Når det nå foreligger to linjer mot målet fra to forskjellige opphavspunkt er det mulig å nøyaktig bestemme målets posisjon. I en ideell verden vil de to linjene møtes i et punkt som blir målets posisjon. Desverre er verden sjeldent så god og det må medregnes at støy og unøyaktigheter gjør at linjene ikke vil krysse. Objektets posisjon kan likevel estimeres ved å finne det punktet som ligger nærmest begge linjene.

Det andre kameraet vil så bruke objektets posisjon i dets bilder til å følge måle videre. Om målet er plassert i øvre del av bildet, kommanderes hodet til å styre kameraet til å peke mer oppover. Dette gjør at objektet kan følges over tid og det produseres kontinuerlige målinger av dets posisjon.

Med dette skulle leseren kunne forstå tanken bak systemets operasjon og sammensetning. Se figur 3.1 for en illustrasjon. I neste delkapittel vil det diskuteres hvordan systemets nøyaktighet og følsomhet for støy ble undersøkt ved hjelp av simulering. Deretter vil valget av de ulike fysiske komponentene diskuteres. Videre vil den spesifikke implementasjonen av systemet presenteres og diskuteres i lys av den programvaren som ble utviklet for å oppnå systemets mål.

3.2 Simulering

Matematisk sett er dette et relativt komplekst system med mange interagerende biter med informasjon som alle påvirker den endelige nøyaktigheten til de målingene systemet utfører. For å undersøke denne sammenhengen og hvordan støy og feilkilder i de forskjellige målingene påvirker systemet har systemet blitt simulert på forhånd. Dette gjør det mulig å se hvilke utslag forskjellige feilkilder gir på en effektiv måte.

Systemet er blitt simulert i python ved bruk av numpy-biblioteket for å gjøre matriseregningen enkel. Her er alle de faktiske posisjoner, vinkler, også videre, kjent nøyaktig og det er mulighet for å legge på feil eller støy på disse av ønsket størrelse. De typene feilkilder som det er fokusert på er feil i vektoren som angir den relative posisjonen til de to kameraene, at de to kameraene er litt rotert i forhold til hverandre og støy på retningsvektorene produsert av de to kameraene. De to første er konstante feil. Det vil si at det antas at denne feilen er like stor hele tiden mens systemet testes. Dette er fordi disse feilkildene skyldes unøyaktigheter i oppsettet av systemet. Under simuleringen ble feilen i kameraposisjon modellert ved å velge en tilfeldig enhetsvektor for så å skalere denne til ønsket lengde. Dette gjøres ved å velge x, y, og z, komponentene ved en normalfordeling rundt null for så å normere vektoren.

Den siste feilkilden, støy på retningsvektorene, derimot antas å være variabel. Denne støyen opptrer i form av en liten endring i retning på den korrekte vektoren og det antas at støyen er forskjellig ved hver måling. I denne simuleringen er støyen modellert på en slik måte at vinkelen mellom den korrekte og støypåvirkede vektoren er satt til å ha en viss størrelse. Videre er aksen denne rotasjonen skjer rundt valgt blant alle dem som står normalt på den originale vektoren med uniform fordeling. Dette medfører at den støypåvirkede vektoren opptrer som den originale vektoren, bare rotert en gitt mengde i en tilfeldig retning.

Disse feil- og støybelagte dataene brukes deretter videre til å estimere objektets posisjon i henhold til fremgangsmåten lagt frem i kapittel 2.1. Dette vil gi målinger som vil ha samme type feil som det som kan forventes av det virkelige systemet og gjør det mulig å undersøke hvilke feilkilder som er mest viktige å kontrollere og holde små. De resulterende estimatene av objektposisjonen ble så analysert ved hjelp av sciPy-biblioteket og fremstilt visuelt ved hjelp av matPlotLib. Resultatene fra disse simuleringene er lagt frem i kapittel 4.1 og diskutert i kapittel 5.1.

3.3 Maskinvare

For å oppnå det systemet som er beskrevet i kapittel 3.1 kreves det en del maskinvare. Det trengs et 360°-kamera, et vanlig kamera, et styrbart hode dette kameraet kan monteres på, samt en datamaskin som kan styre og kommunisere med disse enhetene. Da det er ønskelig at dette systemet skal produsere målinger med høy nøyaktighet kreves utstyr som kan levere dette.

360°-kameraet som er tatt i bruk er FLIRs Ladybug 5+. Dette kameraet er valgt da det ble ansett som det beste ferdige 360°-kameraet som var tilgjengelig ved prosjektets oppstart. FLIR Ladybug 5+tilbyr både høy oppløsning og god bilderate. Det andre kameraet som er tatt i bruk er et FLIR Oryx. Dette er et lite og lett kamera som gjør det ideelt for montering på en styrbar plattform, samtidig som det tar veldig høyoppløste bilder med høy bilderate. Enheten som brukes for å styre dette kameraet mot målet er et FLIR PTU-5. Dette er et pan/tilt hode med svært høy nøyaktighet.

Dette er kun en kort oversikt over den maskinvaren som er valgt til denne oppgaven. I de neste tre delkapittelene skal det gåes mer i dybden på de forskjellige komponentene, hvilke fordeler de har og eventuelle utfordringer de bringer med seg.

3.3.1 Ladybug 5+

360°-kameraet som er blitt tatt i bruk i dette prosjektet er et Ladybug 5+fra FLIR. Dette kameraet består av seks individuelle kameraenheter som peker i hver sin retning, fem rundt kameraets ekvator og et rett opp. Til sammen dekker disse seks enhetene omlag 90% av den synlige sfæren, hvor det som ikke er synlig er det som er rett under kameraet.

FLIR Ladybug 5+

Hver kameraenhet leverer bilder i en oppløsning på 2448×2048 piksler. Dette betyr at det totale bildet fra alle seks sensorene har 30 megapiksler. Disse bildene leveres over USB 3.1 med opptil 30 bilder i sekundet. Oppløsningen kan også halveres for å oppnå 60 bilder i sekundet.

Med 2048 horisontale piksler per kameraenhet og fem enheter rundt kameraets ekvator har kameraet 10240 piksler rundt ekvator. Hver enhet har noe overlapp med de to ved siden av seg og har en horisontal synsvinkel på omlag 80°. Dette gir $\frac{80°}{2048\,\mathrm{p}} = 0,039$ grad per piksel. Dette vil si at på 1 km avstand vil et piksel i bilder fra Ladybugen dekke et område på $\frac{2 \cdot 1000\,\mathrm{m} \cdot \tan(80°/2)}{2048\,\mathrm{p}} = 0,82\,\mathrm{m}$. På 5 km avstand øker dette til omlag 4,1 m.

Dette er ganske mye og viser ulempen med å bruke et kamera med så stort synsfelt, nemlig av det har lav oppløsning per grad av synsfelt. Kameraet kan altså se mye på en gang, men detaljer blir fort for små til å plukkes opp. Flere detaljer om dette kameraet finnes i [5].

Ladybug-kameraet leveres med et programmeringsgrensesnitt kalt Ladybug API som tillater innstilling av kameraet samt å lese bilder fra det via C++ funksjoner. En funksjon som er inkludert i dette grensesnittet skal vise seg å ha stor nytte for systemet. Grensesnittet inneholder nemlig støtte for geometrisk syn og har en funksjon kalt *pixelToRay* som beregner en linje i det kameralokale koordinatsystemet som peker langs retningen til en piksel tilhørende en gitt kameraenhet. Denne linjen skal, ifølge kameraprodusenten, ha en nøyaktighet på ±2 mm på 10 m avstand, eller om lag 0,02 %. Dette vil si at denne linjen ikke skal bomme med mer enn 1 m på 5 km avstand.

3.3.2 Oryx

FLIR Oryx

I forrige delkapittel ble det nevnt noe om Ladybugens lave oppløsning i forhold til dets store synsfelt. For å bøte på dette og få et mer høyoppløselig bilde av målet som skal følges inkluderer systemet et kamera med betraktelig smalere synsfelt og samtidig høy oppløsning. Til dette formålet brukes et Oryx kamera fra FLIR. Mer spesifikt modellen ORX-10G-123S6-C. Dette kameraet tar bilder i 4096×3000 oppløsning og så fort som 68 bilder i sekundet. Hvor bredt synsfeltet til dette kameraet er avhenger av hva slags linse som er installert på det, men ta for eksempel en linse som gir et 50° synsfelt. Da har dette kameraet $\frac{50°}{3000\,\mathrm{p}} = 0,0166$ grader per piksel, altså under halvparten av det Ladybugen har. Dette vil si at en piksel vil dekke et område 31 cm bredt på 1 km avstand og 1,55 m bredt på 5 km avstand. Dette vil gi mye mer detaljerte bilder av målet systemet følger enn det Ladybugen kan produsere. Detaljnivået kan også økes ved å velge en linse med smalere synsfelt.

For å kunne levere disse 12,3 megapiksel bildene i en hastighet på 68 bilder i sekundet kreves en kanal med mer båndbredde enn det USB3 tilbyr. Derfor er Oryx-kameraet utstyrt med et grensesnitt som kommuniserer via 10 Gbit ethernet. Grunnet dette andre fysiske grensesnittet krever også dette kameraet et annet programmeringsgrensesnitt enn Ladybugen. Oryx-kameraet bruker istedet FLIRs Spinnaker API som er bygd for kommunikasjon med kameraer over 10GigE. Det at de to kameraene har forskjellige programmeringsgrensesnitt kompliserer systemet litt, men fordelene med Oryx-kameraet veier opp for dette. Flere av Oryxen tekniske detaljer gjennomgås i [6].

3.3.3 PTU-5

Når det nå er valgt et kamera med smalere synsfelt som kamera nummer to kommer også kravet om å kunne styre dette kameraet mot mål som skal følges. For dette systemet løses dette ved å montere Oryx-kameraet på et styrbart hode som kan både panorere og vinkle seg opp og ned. Et såkalt pan/tilt hode. Mer spesifikt er det tatt i bruk et PTU-5 fra FLIR.

Denne PTUen har en enkoderoppløsning på 0,05°og en nøyaktighet på 0,1°. Denne høye oppløsningen og nøyaktigheten er det som gjør denne PTUen godt egnet til dette prosjektets formål da nøyaktighet er et av systemets krav. Hodet kan også bevege seg så fort som 150°/s side til side og 50°/s opp og ned. Dette gjør at kameraet kan reagere raskt på mål som detekteres av 360°-kameraet.

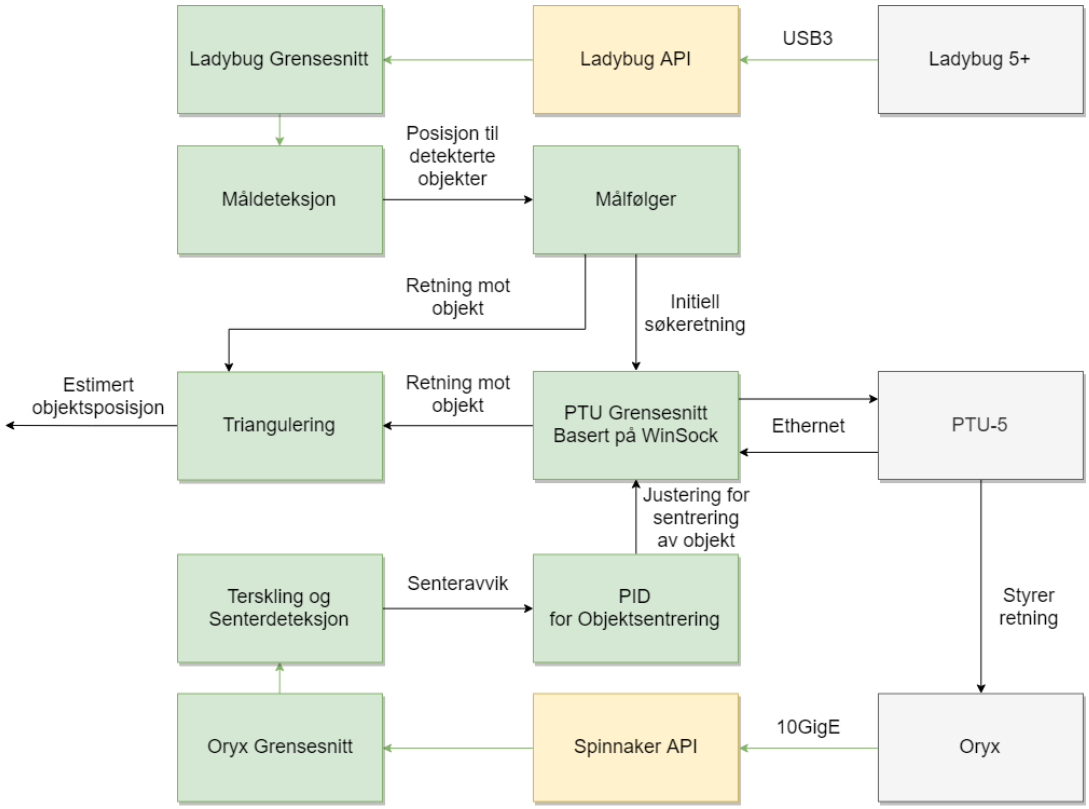
Denne PTUen kan styres enten via RS-485 eller ethernet. Begge disse tilkoblingsmulighetene bruker samme kommandosett og er derfor nesten funksjonelt like. Det som er forskjellen er at for å kunne kommunisere over RS-485 med vertsmaskinen vil en adapter være nødvendig. Flere spesifikasjoner av denne enheten finnes i [4].

FLIR PTU-5

Alle disse komponentene skal altså knyttes sammen til et egenstående system og fungere sammen til å utføre målinger. Denne sammenknyttingen skjer ved at alle komponentene kobles opp mot en datamaskin som kjører både egenlagd programvare og de programmeringsgrensesnittene som er utviklet for de ulike komponentene. Denne maskinen vil benytte kraftige komponenter slik at bildeprosesseringen kan foregå i sanntid. Det er ikke krav om et nøyaktig sett med maskinvare for denne datamaskinen annet enn at den må ha støtte for de grensesnittene kameraene bruker, altså 10GigE og USB3, og kjøre Windows 10. Dette er på grunn av den selvutviklede programvaren som skal kjøres, som kun er utviklet for dette operativsystemet. Denne programvaren og dens virkemåte er tema for neste delkapittel.

3.4 Programvare

Det er frem til nå gitt en enkel overordnet beskrivelse av systemet og en mer grundig beskrivelse av de fysiske komponentene som utgjør systemet. Det er fortsatt mange åpne spørsmål knyttet til hvordan systemet faktisk opererer på et mer detaljert nivå. Å besvare disse spørsålene er formålet med dette delkapitelet. Her vil den programvaren som styrer systemet bli lagt frem. Dette inkluderer i detalj hvordan alle delene av systemet opererer, både individuelt og samspillet mellom dem. Det overordnede designet av programmet er vist i figur 3.2. De grå boksene i diagrammet indikerer maskinvarekomponenter, de gule er grensesnittene fra produsenten, og de grønne er selvlagd programvare. De grønne pilene indikerer at det er bilder som overføres.



Figur 3.2: Systemets informasjonsflyt

3.4.1 Grensesnitt

Som tidligere beskrevet består dette systemet av tre maskinvare-komponenter som må kommuniseres med. Det må være mulig å både lese bilder fra kameraene og konfigurere dem under kjøring. Samtidig må PTUen kunne styres og dens posisjon kunne leses. Det som kompliserer denne prosessen er at de to kameraene har to helt forskjellige grensesnitt. Ladybugkameraet sender bilder over USB 3 og kan styres ved hjelp av utviklerverktøyet Ladybug SDK. Oryxkameraet, på den annen side, krever mer båndbredde en det som er tilgjengelig over USB 3. Dette kameraet bruker i stedet 10 Gbit ethernet som sin kommunikasjonskanal. Dette kameraet kan styres og leses fra via utviklerverktøyet Spinnaker API. Til sist kommer PTU-5 som kan styres via seriell-kommunikasjon eller ethernet. Denne enheten har også et tilhørende utviklerverktøy som det kan aksesseres gjennom.

Felles for disse verktøyene er at de alle støtter tilgang via programmeringsspråket C++ og kompilering via Visual Studio. Dette gjør at et program kan lages som kan kommunisere med alle tre enhetene.

En videre komplikasjon er at dette systemet trenger å utføre ganske omfattende bildeprosessering for å fungere. Dette ville være en utfordring å programmere fra bunn og det er derfor ønskelig å benytte et bibliotek for dette. Valget falt derfor på å inkludere bildeprosesseringsbi-

blioteket OpenCV i prosjektet. Dette biblioteket har åpen kildekode og støtter C++ og Visual Studio.

Begge de to kameragrensesnittene, Ladybug SDK og Spinnaker API, leverer bilder i sine egne formater og kan være kompliserte å bruke. Det ble derfor opprettet egne, selvlagde, grensesnitt mot disse med et mindre utvalg av funksjoner som også gjør om bildene til OpenCV-format. Dette gjemmer vekk en del av den mer komplekse funksjonaliteten disse kameraene har, men denne er ikke nødvendig for dette prosjektets formål. Det gjør også at hovedapplikasjonen ikke trenger å ta høyde for den spesifikke implementasjonen hvert kamera krever og øker brukervennligheten.

Videre har det vist seg strevsomt å få FLIRs PTU-SDK til å fungere som den skal. Valget falt derfor på å lage et eget grensesnitt for å styre PTUen. Dette gjøres ved at PTUen kobles til via ethernet og så sendes kommandoer via HTTP POST-forespørsler. Disse forespørslene sendes via C++ ved hjelp av WinSock biblioteket som er innebygd på Windows. Dette har den fordelen at et veldig enkelt og brukervennlig grensesnitt er blitt laget, som er skreddersydd for kravene tilknyttet akkurat dette bruksområdet, uten at programmet trenger å kompilere inn flere tredjeparts-biblioteker. Ved å styre over ethernet og ikke RS-485 er det heller ikke nødvendig med noen adapter som gjør oppsettet litt enklere.

3.4.2 Ladybug Måldeteksjon

Som nevnt er det ene kameraet i dette systemet et 360°-kamera som står statisk på bakken og kan se i alle retninger. Dette kameraet skal kunne oppdage når et fly eller et annet flyvende objekt gjør seg synlig på himmelen for så å beregne en linje i rommet dette objektet befinner seg langs. Denne delen av systemet har ved oppstart ingen informasjon om hvor objektet muligens befinner seg, annet at det er på himmelen. Det som derimot kan rimelig antas er at objektet vil være i bevegelse. Da kameraet i seg selv er statisk er det mulig å utføre forgrunnsdeteksjon på bildene det produserer for å lokalisere målene.

Som nevnt i kapittel 3.4.1 benytter systemet et grensesnitt mot kameraet som gjør at bilder kan leveres i OpenCV-format. Dette er praktisk da OpenCV har innebygget støtte for flere typer forgrunnsdeteksjon. Det må medregnes at bakgrunnen i bildene fra Ladybugen ikke er fullstendig statiske, da de inneholder ting som trær, skyer, gress og kanskje vann som vil oppleves som bevegelse i bildet. Det er ønskelig at forgrunnsdeteksjonsalgoritmen tar høyde for dette og markerer dette som bakgrunn tross de små bevegelsene. Dette oppnås best ved hjelp av en MOG-basert metode, som beskrevet i kapittel 2.2.

Valget falt derfor på OpenCVs *BackgroundSubtractorMOG2*. Dette er en meget robust, adaptiv forgrunnsdeteksjonsalgoritme som er basert på [9] og [8]. Fordelen med denne MOG-algoritmen er at den adaptivt velger antallet normalfordelinger som kreves for hvert piksel, heller enn å bruke et fast antall på alle. Dette gjør at de delene av bildet som oppleves nesten helt statisk, som den blå himmelen, kun behøver én fordeling. De mer komplekse delene, som kantene rundt skyer, kan derimot ha flere. Dette gjør at god nøyaktighet kan oppnås på de komplekse delene samtidig som det sparer ressurser ved å kun bruke flere fordelinger der det trengs.

Funksjonaliteten til *BackgroundSubtractorMOG2* må konfigureres til systemets formål. Dette gjøres via de forskjellige parameterene algoritmen bruker. Disse parameterene er *history*, *varThreshold* og *detectShadows*. *DetectShadows* er den enkleste å gjøre rede for da denne ikke brukes i dette systemet. Denne parameteren gjør at forgrunnsdetektoren også klassifiserer skygger av forgrunnselementer som en egen klasse i bildene den prosesserer. Dette har ingen nytte for dette prosjektets bruk av algoritmen og denne parameteren settes derfor til *false*.

De to andre parameterene forgrunnsdetektoren bruker er litt vanskeligere å velge. *History*-parameteren avgjør hvor mange bilder bakover i tid algoritmen bruker for å danne bakgrunnsmodellen. Valget av denne parameteren krever en viss balanse for at dette systemet skal fungere slik det skal. Det er klart at med en lengre historie vil algoritmen klare å bygge opp en mer korrekt bakgrunnsmodell og ha større tillitt til denne modellen. Dette høres i utgangspunktet ut som utelukkende en fordel, men det er også en ulempe. Hvis bakgrunnsmodellen algoritmen bruker er svært nøyaktig, trengs det veldig lite for å gi utslag som forgrunn. Dette viste seg å bli et problem under testing av systemet, i hovedsak på grunn av skyer. Skyer er noe dette systemet må kunne tåle å ha i bakgrunnen når det opererer, men det er ikke ønskelig at disse skal markeres som forgrunn. Likevel er skyer objekter som beveger seg over himmelen, bare svært sakte. Om *history*-parameteren velges for høy vil skyer gi utslag som forgrunn. Likevel er det ønskelig å holde parameteren relativt høy da dette tillater systemet å oppdage objekter som er meget små, selv så små at de kun utgjør et par piksler. Dette er nyttig da det tillater systemet å oppdage objekter fra større avstand og dermed tidligere.

Den siste parameteren som må velges er *varThreshold*-parameteren. Denne parameteren bestemmer enkelt nok hvor mye et nylig observert piksel må skille seg fra bakgrunnsmodellen for å markeres som forgrunn. Her oppstår igjen noe av samme problemet som med *history*-parameteren. Det er ønskelig at også små objekter skal kunne detekteres av systemet og det er derfor ønskelig å holde *varThreshold* lav. Ulempen med dette er at det også åpner systemet for flere feilaktige deteksjoner hvor en litt dynamisk bakgrunn markeres som forgrunn.

Disse variablene må altså konfigureres før systemet tas i bruk. Det kan antas at scenen er svært lik hver gang systemet skal bruke. Det burde derfor ikke være nødvendig med betydelige re-kalibreringer mellom hver gang systemet brukes. En kalibrering av disse parameterene som fungerer for en utendørsscene vil fungere fint på andre lignende scener.

Figur 3.3 viser steg for steg hvordan denne delen av systemet fungerer. I figur 3.3d er det mulig å se at systemet har plukket opp tre objekter. Disse er dronen, dronens skygge og en fugl som flyr gjennom scenen. I tillegg er det mulig å skimte litt støy fra gresset nede til høyre i bildet. Algoritmen vil over tid kunne skille disse objektene fra hverande basert på deres oppførsel. Skyggen forsvinner så fort dronen flyr ut over gresset, og er dessuten for langt nede i bildet til å være et flyvende objekt. Gresset vil bare oppdages i korte perioder når det plutselig beveger seg mer enn vanlig, og er også for langt nede. Fuglen flyr alt for fort til å kunne være objektet som skal følges. Da gjenstår kun dronen som er det målet det er ønskelig å følge.

Systemet har nå produsert en maske som angir hvilke piksler av bildet utgjør forgrunnsobjekter og hvilke som er av bakgrunn. For videre arbeid er det ønskelig at de pikslene som henger sammen betraktes som ett objekt. Dette utføres ved hjelp av OpenCVs *findContours*-funksjon. Denne funksjonen grupperer piksler som henger sammen og returnerer dem som en liste med lister av piksler. Hver av disse listene med piksler er da alle pikslene som henger sammen og danner et sammenhengende objekt. Da dette er mye data som ikke er nødvendig for dette prosjektets bruk gjøres dette om til en datastruktur som kun inneholder to datapunkter. Disse datapunktene er objektets sentroide, altså den gjennomsnittlige posisjonen til alle pikslene som utgjør objektet, og et minimalt innskrivingsrektangel, altså det minste rektanglet som fullstendig omkranser objektet.

Denne delen av systemet analyserer bildene fra Ladybug-kameraet og returnerer en liste med forgrunnsobjekter, beskrevet ved sentroide og innskrivingsrektangel. Videre er det ønskelig at bevegelsen til disse objektene over tid følges gjennom bildeplanet. Hvordan dette oppnås vil diskuteres i neste delkapittel.

(a) Bilde sendt inn til algoritmen

(b) Bakgrunnsmodellen

(c) Forskjell mellom bilde og bakgrunn, tersklet

(d) De individuelle objektene segmentert

Figur 3.3: Eksempel på forgrunnsdeteksjon

3.4.3 Ladybug Målfølging

Systemet har nå produsert en liste med punkter og rektangler som beskriver forgrunnsobjekter i et bilde fra Ladybug-kameraet. Systemet skal i utgangspunktet følge og posisjonere ett mål, men det er rimelig å anta at noen feildeteksjoner gjøres av måldeteksjons-systemet. Derfor må de detekterte målene følges over flere bilder slik at systemet får mer informasjon om dem. Systemet kan da evaluere hvilke mål som har vært konsekvent synlig over lengre tid og hvilke mål som har en rettlinjet bevegelse. Det målet som scorer høyest på disse faktorene er mest sannsynlig det målet hvis posisjon er av interesse.

Systemet vil derfor holde en liste med objektposisjoner som det kan oppdatere når et nytt bilde kommer inn fra Ladybugen. Denne listen vil inneholde hvor objektet befinner seg nå, samt dets innskrivingsrektangel og en historie med tidligere posisjoner. Ut i fra denne listen er det mulig å bestemme hvilket objekt som mest sannsynlig er det interessante målet.

For å lage og oppdatere denne listen er det nødvendig at systemet kan matche tidligere detekterte objekter med nye deteksjoner fra et nytt bilde. Systemet må velge disse matchene på en slik måte at samme fysiske objekt kun tilsvarer ett objekt i lista og at to objekter ikke bytter følge. Måten systemet hånderer dette på er ved å kalkulere en differanse-score mellom hvert tidligere detekterte objekt og hvert nye detekterte objekt. Denne differanse-scoren baserer seg på avstanden mellom sentroidene på de to objektene samt forskjellen i høyde og bredde på innskrivingsrektanglene. Disse tallene skaleres også med størrelsen til objektene for at de skal være relative.

Etterhvert som systemet regner ut de forskjellige differanse-scorene legges disse inn i en prioritetskø. Dette gjør det enkelt å hente ut den matchen med lavest differanse. Matchen med lavest differanse hentes ut om og om igjen helt til alle objekter har fått en match eller alle de gjenværende differansene er høyere enn en satt terskelverdi. Objekter som har blitt fulgt før, men som ikke har fått en match i dette bildet får en variabel inkrementert. Har ikke objektet fått noen match i flere påfølgende bilder slettes dette objektet fra listen over fulgte objekter. Nye objekter som ikke har fått noen match med tidligere deteksjoner legges til i lista. Se algoritme 1 i appendix.

En annen ting som dette systemet må ta høyde for er objekter som forsvinner i kortere perioder. Ta eksempelet med et fly som følges over himmelen før det forsvinner bak en sky. Etter en liten stund dukker flyet opp på den andre siden av skyen. Mens flyet er bak skyen vil det ikke detekteres av måldeteksjons-algoritmen, likevel er det ønskelig at systemet forstår at objektet som dukker opp på den andre siden av skyen er det samme som forsvant bak den for litt siden. Dette kan systemet oppnå ved å implementere et kalmanfilter for de fulgte objektene.

Ved å lage kalmanfilteret slik at det antar at objekter har konstant fart i bildeplanet vil filteret kunne gi oss en prediksjon på hvor et objekt kommer til å detekteres i neste bilde gitt at det har blitt detektert et par ganger tidligere. Hvis systemet da bruker denne prediksjonen når nydetekterte objekter skal matches med de som allerede følges vil vi få enda bedre målfølging. Da vil systemet kunne predikere at flyet som forsvant bak skyen for en viss tid siden vil dukke opp igjen omtrent på andre siden av skyen. Når flyet da dukker opp igjen på dette punktet vil systemet kunne forstå at dette nye objektet faktisk er det gamle objektet som bare ikke ble detektert på en stund.

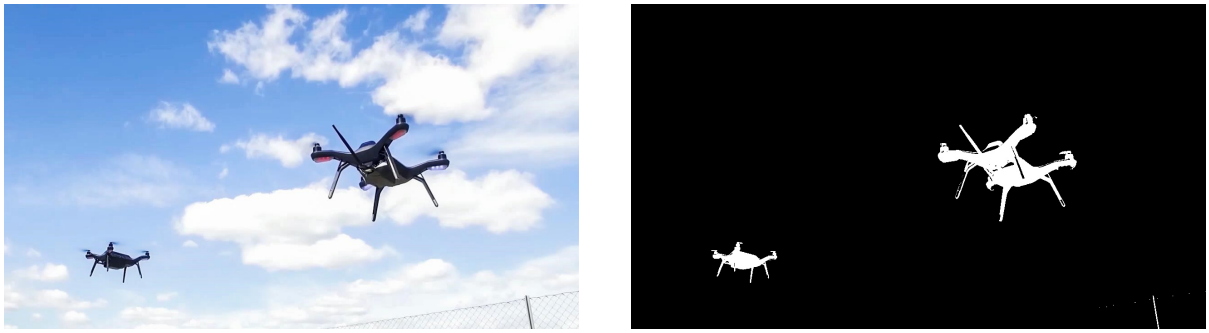
Systemet har nå muligheten til å detektere forgrunnsobjekter i bildene fra Ladybug-kameraet og følge disse objektene i deres bevegelse over bildeplanet, selv om det tidvis ikke blir detektert. Det neste steget blir å finne det samme målet med Oryx-kameraet og finne de to linjene som peker mot objektet. Med denne informasjonen vil systemet være istand til å angi objektenes posisjon.

3.4.4 Oryx og PTU-5 Målsøk og Målfølging

Fra målsøket som er gjort i bildene fra Ladybugen foreligger det nå en tredimensjonal linje i rommet som peker fra Ladybugen mot målet hvis posisjon skal estimeres. For å kunne bestemme denne posisjonen trengs det enda en linje som peker mot målet, fra et annet sted enn Ladybugen. Å finne denne linjen er oppgaven til Oryx-kameraet og PTUen det er montert på. Da Oryxen er montert på en PTU som skal peke kameraet i forskjellige retninger holder åpenbart ikke kravet om et statisk kamera som er nødvendig for å kunne bruke forgrunnsdeteksjonsalgoritmen som ble brukt på Ladybugens bilder. Derimot er objektets posisjon nå mye mer begrenset enn det var for Ladybugen da det er gitt en linje objektet ligger langs. Dette gjør at det ikke er nødvendig å bruke en så sofistikert metode som forgrunnsdeteksjon. Istedet kan noen antagelser om objektet og scenen brukes til å gjøre dette målsøket betydelig enklere.

Det er ikke urimelig å anta at objektet som skal lokaliseres er mørkere enn dens bakgrunn. Det er heller ikke urimelig å anta at hele objektet skal kunne få plass på et enkelt bilde fra Oryxen. Fremgangsmåten for å lokalisere objektet blir da å søke langs linjen fra Ladybugen til et objekt som oppfyller disse kravene oppdages. Det er usannsynlig at dette skulle finne andre objekter enn det som følges av Ladybugen, men selv om dette skulle være tilfellet vil dette enkelt oppdages når trianguleringen utføres. Om kameraet har funnet et annet objekt enn det Ladybugen følger vil trianguleringen vise at det er stor avstand mellom de to linjene etter at målet Ladybugen følger har beveget seg litt. Da kan målsøket for Oryxen gjenopptas. Dette kan gjøres helt til riktig mål er lokalisert. I figur 3.4 vises et eksempel på hvordan terskling kan detektere mørke objekter mot en lys himmel. I figur 3.5b vises det at gjerdet i dette bildet også

betegnes som et mulig mål. Dette vil kunne forvirre algoritmen, men vil ikke være et problem for mål som befinner seg høyere oppe. Som nevnt vil også trianguleringen kunne vise at dette ikke er riktig mål.



(a) Mørke droner på lys bakgrunn **(b)** Det samme bildet etter terskling

Figur 3.4: Eksempel på terskling av mørke objekter på lys himmel

Når Oryxen nå har funnet det riktige målet har denne delen av systemet to oppgaver. For det første må kameraet fortsette å følge målet. Dette gjøres først ved at objektets sentroide blir kalkulert. Deretter justeres vinklene på PTUen slik at sentroiden blir sentrert i bildet fra Oryxen. Altså om sentroiden er i øvre venstre del av bildet justeres kameraet til å peke litt mer opp og mot venstre. Dette kan utføres med en enkel todimensjonal PID regulator. Når denne oppdateres flere ganger i sekundet vil alltid kameraet peke direkte mot objektet.

Systemets andre oppgave er å definere linjen som peker mot objektet. Ved å benytte det at objektet som skal posisjoneres ligger i sentrum av bildet er dette så enkelt som å finne kameraets optiske akse. Dette gjør at de interne parameterene av kameraet ikke trenger å være kjent for å kunne produsere en linje mot objektet. I stedet holder det å vite kameraets posisjon og i hvilken retning det peker.

For å finne disse to bitene med nødvendig informasjon holder det å se på PTUens oppbygging og vinkler. For dette prosjektet har Oryxen blitt montert på vestre side av PTUen og dens høyde er slik at kameraets senter krysser rotasjonsaksen for PTUens tilt. Definer at x-aksen peker i samme retning som PTUens 0° pan. Da er et punkt på kameraets optiske akse gitt ved sylinder-koordinatene $\mathbf{c} = [r, \theta, h]$, hvor r er avstanden mellom PTUens pan akse og kameraets senter, θ er PTUens pan-vinkel, og h er høyden PTUens tilt-akse er over basen. Om basens posisjon er kjent vil da kameraets posisjon være mulig å regne ut.

Videre må det produseres en enhetsvektor som peker i samme retning som kameraet. Dette gjøres enkelt ved å bruke sfæriske koordinater. Enhetsvektoren $\mathbf{u}$ kan da beskrives ved $\mathbf{u} = [1, \theta, \phi]$ hvor 1 er radiusen, denne settes til 1 for å få en enhetsvektor, θ er PTUens pan-vinkel

og ϕ er tilt-vinklen. I kartesiske koordinater kan disse skrives som

$$\mathbf{c} = \begin{bmatrix} r \cdot \cos(\theta) \\ r \cdot \sin(\theta) \\ z \end{bmatrix} \quad \text{og} \quad \mathbf{u} = \begin{bmatrix} \cos(\theta)\cos(\phi) \\ \sin(\theta)\cos(\phi) \\ \sin(\phi) \end{bmatrix}. \tag{3.1}$$

Når disse to bitene med informasjon settes sammen vil det danne en linje som peker fra Oryxens senter mot objektet. Dette sammen med informasjonen som er kjent fra før, linjen fra Ladybugen mot målet og den relative posisjonen mellom de to kameraene, er nok til å fastslå objektets posisjon. Utførelsen av dette er beskrevet i neste delkapittel.

3.4.5 Triangulering og Datasmelting

Med dette skulle all informasjonen som er nødvendig for å kalkulere objektets posisjon være funnet. Det eneste som da gjenstår er å beregne objektets posisjon slik det er beskrevet i kapittel 2.1. Dette ble implementert via OpenCV da dette biblioteket har støtte for matriseoprasjoner som multiplikasjon, invertering og transponering.

Dette regner altså ut objektets posisjon samt den minste oppnåelige avstanden mellom de to linjene som trianguleringen ble utført på. Om denne minste avstanden viser seg å være stor, er det rimelig å anta at de to linjene ikke egentlig peker på samme objekt og målsøket for Oryxen må startes på nytt.

For å optimalisere flyten i programmet og utnytte alle ressursene i datamaskinen som styrer systemet optimalt ble det forsøkt å parallellisere innhentingen og prosesseringen av data i systemet. Det ble opprettet en tråd for hver av de tre oppgavene. En for å innhente bilde fra Ladybugen og oppdatere målfølgeren, en for å innhente bilde fra Oryxen og justere kameravinkelen for å sentrere det fulgte objektet, og en siste for å triangulere posisjonen basert på den nyeste tilgjengelige dataen fra begge kameraene.

Dette ville gjøre at systemet kunne optimalisert bruken av datamaskinens ressurser. For eksempel kan bildet fra Oryxen lastes ned og prosesseres samtidig som målfølgeren til Ladybugen oppdateres da disse prosessene bruker forskjellige komponenter i maskinen. Dette viste seg å ikke fungere i praksis da de medfølgene SDKene fra FLIR ikke er parallelliserbare. Ved å la systemet innhente bilder fra Ladybugen samtidig som bilder ble lastet inn fra Oryxen ble deler av dataen i bildene fra Oryxen på et eller annet vis overskrevet eller korruptert. Bildene kunne da ikke brukes. Den planlagte parallelliseringen måtte derfor henlegges og systemet ble stående som serielt. Altså vil systemet først innhente og prosessere bilder fra Ladybugen for så å gjøre det samme med Oryxen.

Dette er de forskjellige delene med programvare som er blitt satt sammen for å oppnå systemets ønskede funksjon. Det er mange deler og for at systemet skal være i stand til å produsere nøyaktige målinger kreves det at alle sammen fungerer. Nettopp derfor er det brukt biblioteker fra produsenten og velkjente åpne kildekode-bibliotek som OpenCV, og det er benyttet robuste algoritmer som skal kunne operere selv under noe uforutsette forhold. Det er også gjort noen antagelser som er viktige å tenke på når systemet skal settes opp for bruk. Dette er tema for neste delkapittel.

3.5 Oppsett

Når dette systemets skal settes opp er det flere ting som må gjøres riktig og nøyaktig for å sikre at systemet produserer pålitelige målinger. For det første må den relative posisjonen og rotasjonen mellom de to kameraene kontrolleres nøyaktig. Hvis den relative posisjonen mellom kameraene er unøyaktig vil trianguleringen foregå på bakgrunn av unøyaktig informasjon, noe som åpenbart vil gi unøyaktigheter i resultatene den produserer. Det som kanskje er enda viktigere er at den relative rotasjonen mellom de lokale koordinatsystemene til de to kameraene er kjent med høy nøyaktighet. Påvirkningskraften til disse to feilkildene er belyst i kapittel 4.1 og 5.1. Et siste punkt som er viktig å kontrollere er tidsstempling av informasjon. Dette er viktig da systemet bruker to uavhengig produserte biter med informasjon til å produsere målingene. Hvis systemet bruker to målinger som er gjort på forskjellige tidspunkt vil dette åpenbart føre til feil i posisjonsestimatene.

3.5.1 Posisjonering av Kameraene

For at systemet skal fungere optimalt er det viktig å tenke gjennom hvor de to kameraene plasseres når systemet skal tas i bruk. For det første er det viktig at de to posisjonene er kjent relativt til hverandre. Dette må måles nøyaktig før systemet settes igang. I forsøk gjort i dette prosjektet er det benyttet en differensiell GPS (DGPS) for å fastslå posisjonen til hvert kamera med cm nøyaktighet.

Videre er det viktig å tenke på hvor kameraene skal plasseres i forhold til mål som skal følges. Avstanden mellom kameraene en viktig faktor for å sikre gode estimater i trianguleringsfasen. Ligning 2.5 viser at nøyaktigheten til avstandsestimatene som gjøres av trianguleringsalgoritmen øker med kvadratet av sinus til vinkelen som spennes av målet og de to kameraene. Nøyaktigheten blir da høyest hvis de to linjene det trianguleres ved er nermest mulig 90° på hverandre. Hvis målet som skal posisjoneres er 5 km unna kameraene og kamerane bare står for eksempel rundt 20 m fra hverandre blir de to linjene nesten parallelle. Dette gjør at en veldig

liten feil på en av retningene kan få et utrolig stort utslag på posisjonsestimatet. Dette gjør også at kameraenes posisjon i forhold til målet spiller en stor rolle. Hvis målet ligger omtrent på linjen mellom de to kameraene blir også de tilsvarende synslinjene nesten parallelle, uavhengig av avstanden mellom kameraene. Derfor bør kameraene settes opp slik at området der objektet forventes å være ligger langt unna linjen som går mellom de to kameraene. Mer om effekten av kameraavstand og objektets posisjon relativt til linjen mellom kameraene finnes i kapittel 4.1 og 5.1

3.5.2 Orientering av Kameraene

En annen ting som må tas hensyn til under oppsettet av plattformen er kameraenes relative orientering. Det er tatt utgangspunkt i at de to kameraene ikke er rotert i forhold til hverandre, men dette er ikke realistisk å få til med manuelt oppsett. Derfor må det finnes en nøyaktig beskrivelse av den relative orienteringen av de to kameraene slik at dette kan korrigeres for. På denne måten får systemet to retningsvektorer definert i et felles koordinatsystem.

For Ladybugen er dens koordinatsystem definert ved at x-aksen peker ut av kameraenhet 0, z-aksen peker rett opp av kameraenhet 5 (den som peker oppover), og y-aksen velges slik at dette blir et høyrehånds-system. Dette er viktig da LadybugAPIets funksjoner for geometrisk syn, som dette prosjektet benytter seg av, bruker dette koordinatsystemet.

Oryxen, og PTUens koordinatsystem er definert ved at x-aksen peker i retningen som tilsvarer 0 i både pan og tilt vinkel, z-aksen peker i retningen som tilsvarer 90° i tilt vinkel, og y-aksen velges slik at dette blir et høyrehånds-system.

Strålene de to kameraene produserer er definert i sitt tilsvarende kameralokale koordinatsystem. Før målets posisjon kan trianguleres fra disse strålene må de begge transformeres til et felles koordinatsystem. Dette kan gjøres ved å definere et lokalt ENU-system. I dette systemet tilsvarer x-aksen øst, y-aksen nord, og z-aksen rett opp slik at det blir et høyrehånds-system. Origo av dette koordinatsystemet kan plasseres vilkårlig.

Et punkt som er gitt i globale koordinater med høyde, lengde- og breddegrader kan da enkelt omgjøres til et punkt definert i ENU-systemet ved å betrakte forskjellen mellom punktet og origos lengde- og breddegrad samt høyde. For å gjøre dette antas det at øst og nord er samme retning for punktet som for origo. Dette vil ikke stemme nøyaktig på grunn av jordens buethet, men forskjellene er svært små når avstanden mellom punktet og origo kun er noen få kilometer slik som for dette systemet. Det må også gjøres en antagelse om hvor mange grader som tilsvarer en meter, slik at ENU-systemet kan defineres i meter. Dette finnes det heldigvis gode tall på.

Den matematiske definisjonen på omgjørelsen fra globale koordinater til et lokalt ENU-system kan beskrives med følgende. Gitt $p_g = [p_{lat}, p_{lon}, p_{alt}]$ og $o_g = [o_{lat}, o_{lon}, o_{alt}]$ der p_g er et punkt med globale koordinater breddegrad, lengdegrad, og høyde gitt av henholdsvis $p_{lat}, p_{lon}, p_{alt}$ og o_g er ENU-systemets origo, også gitt i globale koordinater. Da kan punktets posisjon i det lokale ENU-systemet beskrives ved $p_{ENU} = [p_x, p_y, p_z]$ hvor

$$p_x = (p_{lon} - o_{lon}) \cdot 111320 \cdot \cos(o_{lat}), \tag{3.2}$$

$$p_y = (p_{lat} - o_{lat}) \cdot 110574, \tag{3.3}$$

$$p_z = p_{alt} - o_{alt}. \tag{3.4}$$

Etter at kameraene er plassert i henhold til de retningslinjer som er lagt frem i kapittel 3.5.1 og deres posisjon er nøyaktig målt med differensiell GPS er det altså mulig å beskrive deres posisjon i ENU-systemet. For å gjøre denne transformasjonen komplett kreves det også at rotasjonen mellom de kameralokale systemene og ENU-systemet.

Denne rotasjonen kan beregnes ved å måle posisjonene til to punkter, som begge er synlige for de to kameraene, med DGPS. Finn så igjen disse punktene i bildene fra kameraene og regn ut linjer mot disse punktene på samme måte som gjøres med mål som skal posisjoneres. Da foreligger to stråler, definert som et utgangspunkt og en retning, definert i det kameralokale koordinatsystemet. Oppgaven blir da å finne en rotasjon av det kameralokale systemet som gjør at de to målte punktene i ENU-systemet ligger langs disse strålene etter rotasjonen.

Det at disse strålene ikke nødvendigvis har samme utgangspunkt gjør det å finne denne rotasjonen vanskelig. Det må også regnes med at det er små unøyaktigheter her så det er ikke gitt at en eksakt analytisk løsning eksisterer.

For å forenkle dette problemet er det derfor her valgt å anse strålene som å ha et felles utgangspunkt i punktet som ligger midt mellom de to faktiske utgangspunktene. Dette er en forenkling som vil bringe med seg noe unøyaktighet i kalibreringen av kamerarotasjonen. Likevel vil alltid disse utgangspunktene ligge nært origo av det kameralokale systemet og det antas at feilen som følge av dette er svært liten.

Den nøyaktige fremgangsmåten for hvordan den beste rotasjonen finnes etter at denne forenklingen er gjort finnes i algoritme 2 i appendix. Denne rotasjonen bestemmes så for hvert kamera og påføres alle målinger som produseres av det tilhørende kameraet før dataen brukes til å triangulere objektets posisjon. Dette sikrer at de strålene som trianguleres med begge er definert i samme koordinatsystem.

3.5.3 Tidsstempling av Informasjon

Til sist må det tas høyde for at dette er et distribuert, diskret system. Målingene systemet gjør, i form av bilder, er tilknyttet spesifikke tidspunkt. Det er viktig at systemet bruker to målinger fra omtrent samme tidspunkt når det fulgte objektets posisjon skal trianguleres. For å oppnå dette sammenlignes tidsstemplene på de bildene som tas.

Ladybugen har støtte for å koble til en GPS via GPIO. Dette gjør at kameraet kan knytte NMEA-meldinger til bildene den leverer som inneholder et tidsstempel med nøyaktighet på omlag 2 ms ved bruk av programvareutløser slik som i dette prosjektet. Oryx på den annen side bruker IEEE 588 tidsprotokoll, som gir mikrosekund nøyaktighet. Dette gjør at systemet kan sammenligne tidsstemplene på bildene fra hvert kamera og utføre trianguleringen med data fra to bilder som er tatt omtrent samtidig.

Dette kapittelet har gitt en gjennomgang av systemets tanke og oppbygging. Det som gjenstår nå er å utføre tester og undersøkelser for å se hvorvidt systemet fungerer i praksis. Hvordan disse undersøkelsene har blitt gjort, og hva de har vist, vil være tema for neste kapittel.

Kapittel 4

Resultater

Etter det foregående kapittelet bør leseren ha opparbeidet seg en god forståelse av systemets virkemåte, både dens grunnleggende tanke og spesifikke implementasjon. I dette kapittelet vil det legges frem hva som er blitt gjort for å teste systemets funksjonalitet og hva disse testene har vist. Verdien og meningen av disse resultatene vil så diskuteres i neste kapittel.

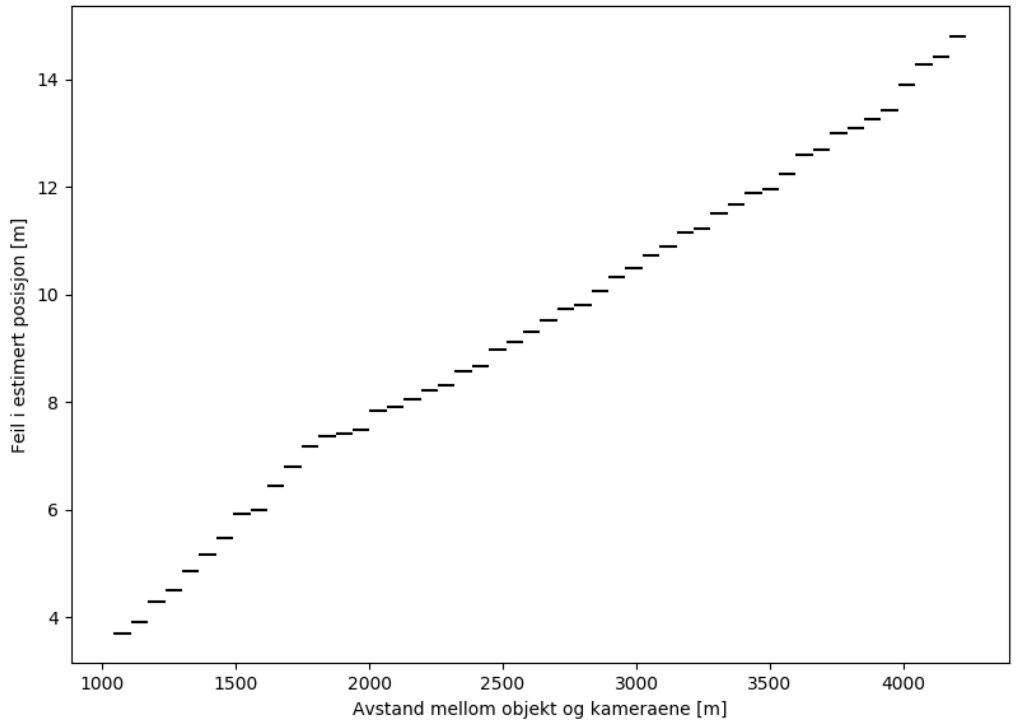
4.1 Simulering

Tidlig i arbeidet ble det laget en simulator av systemet for å teste utslagskraften på de forskjellige typene støy og feilkilder det er rimelig å anta at systemet vil bli utsatt for. Her vil de simuleringene som er gjort og deres resultat legges frem.

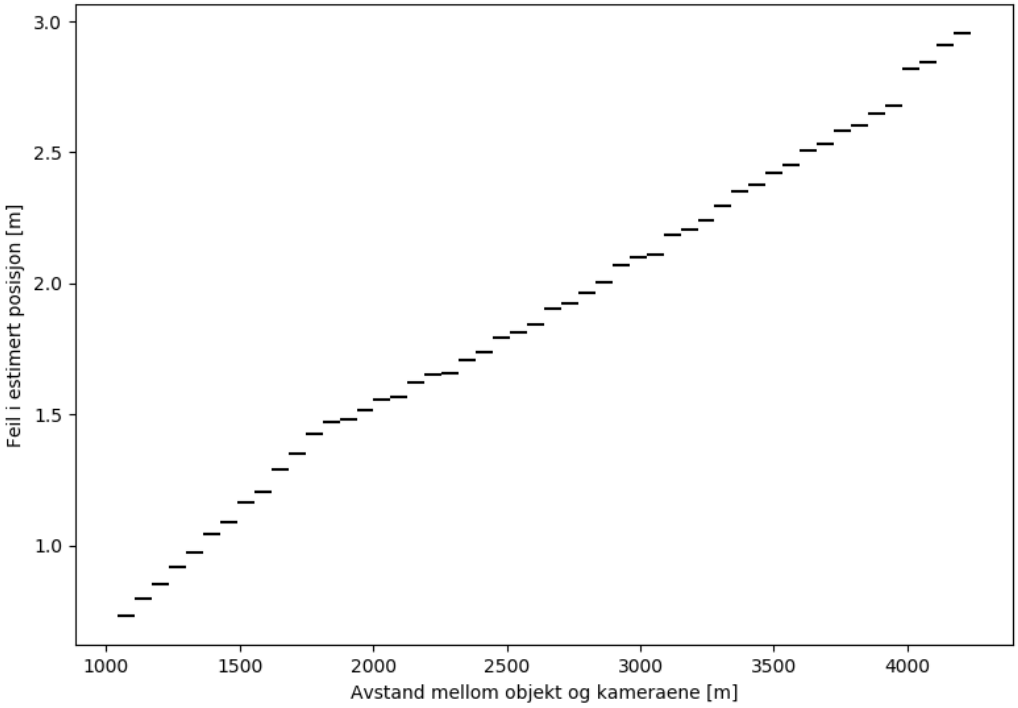
4.1.1 Avvik i Kameraposisjon

Den første testen som ble gjennomført hadde til hensikt å vise hvordan feil i den målte posisjonen mellom de to kameraene ved oppsett ville påvirke den estimerte posisjonen til objekter. I praksis vil denne feilen være konstant under bruk av systemet, men for å få mange verdier ble det valgt 200 tilfeldige avvik for hver posisjon av objektet. Avviket ble satt til å ha en størrelse på 50 cm og påvirker den antatte posisjonen til begge kameraene. For denne simuleringen er objektsposisjonen valgt til å variere et område på 3×3 km foran kameraene. Posisjoner i dette området ble iterert over med 50 m mellom hver posisjon og en høyde 300 m over kameraene. Dette er for å gi en god variasjon i avstand mellom objekt og kamera under testen, samt variere objektets posisjon i forhold til kameraene.

De resulterende posisjonsestimatene ble så fordelt inn i 50 diskrete områder basert på hvor langt objektets faktiske posisjon var fra kameraene for hvert estimat. Figur 4.1 viser den gjennomsnittlige normen av feilen i estimert objektposisjon for hver av disse områdene med en avstand på 50 m mellom kameraene. Figur 4.2 viser samme simuleringen med 500 m mellom kameraene.



Figur 4.1: Feil i estimert posisjon som følge av feil i antatt relativ posisjon mellom kameraene, som funksjon av avstand mellom objekt og kamera. Kameraavstand 50 m, feilutslag 50 cm

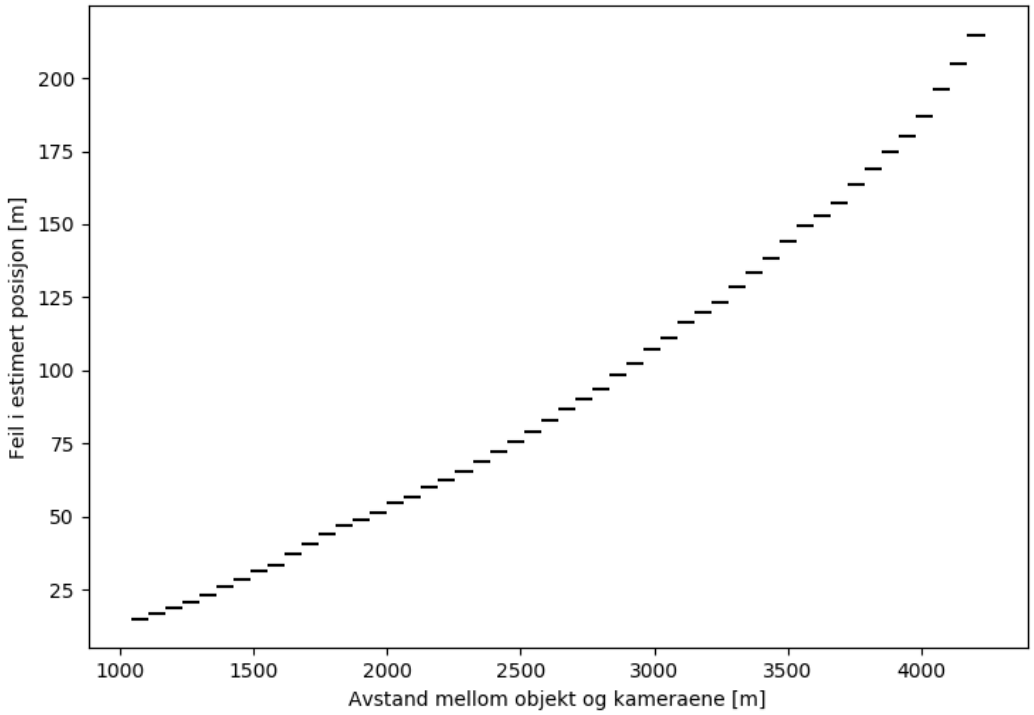


Figur 4.2: Feil i estimert posisjon som følge av feil i antatt relativ posisjon mellom kameraene, som funksjon av avstand mellom objekt og kamera. Kameraavstand 500 m, feilutslag 50 cm

4.1.2 Avvik i Kamerarotasjon

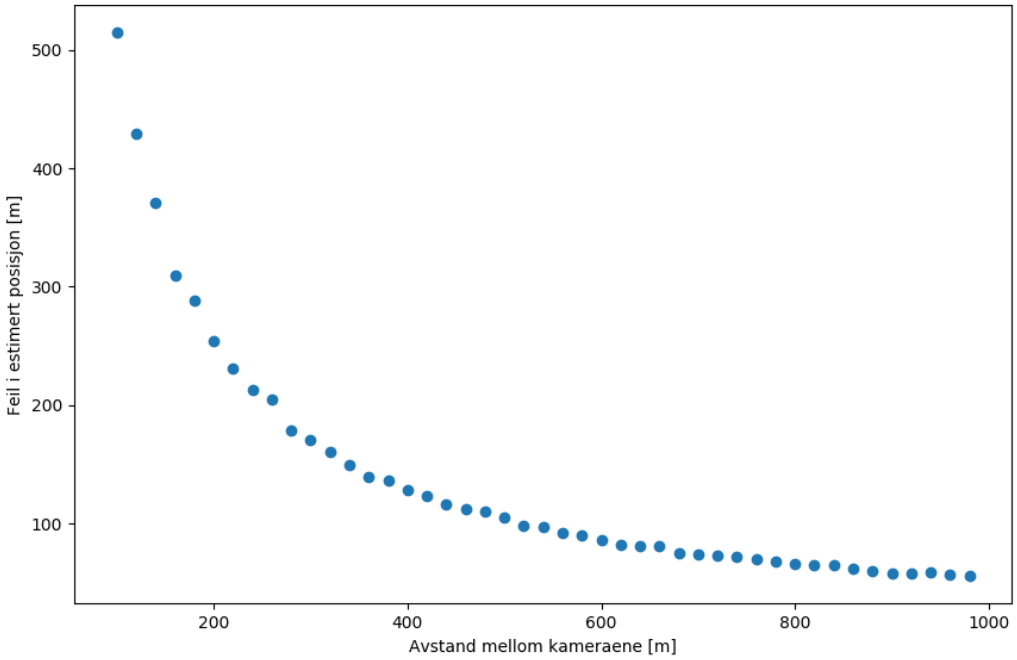
Neste test gikk ut på å analysere påvirkningen av avvik i relativ kamerarotasjon under oppsett. I likhet med testen av kameraposisjon er dette en feil som antas å være konstant under hele tiden systemet er i bruk. Likevel er det ønskelig å gjøre flere målinger slik at hele utfallsrommet for denne feilen blir testet. Denne feilkilden har blitt testet i tre forskjellige sammenhenger. Den første er objektets avstand fra kameraene, den andre er kameraenes avstand fra hverandre, og den siste er utslag i feilvinkel.

I den første testen ble kameraene plassert med 500 m mellomrom og objektets posisjoner ble valgt på samme måte som beskrevet i kapittel 4.1.1. Det ble generert 200 tilfeldige rotasjonsavvik for hver objektposisjon og hver av disse medførte en rotasjon på 0,5°. Denne feilen ble kun påført den ene av de to kameraene. De resulterende posisjonsestimatene ble kvantisert på samme måte som beskrevet i kapittel 4.1.1 og den gjennomsnittlige normen av feilen vises i figur 4.3



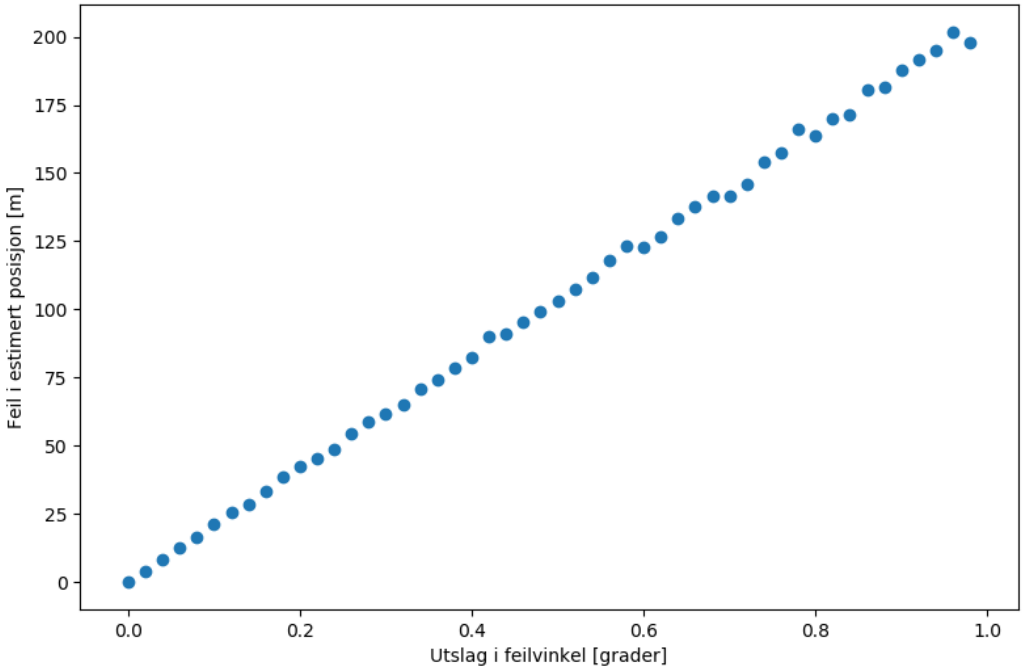
Figur 4.3: Feil i estimert posisjon som følge av relativ rotasjon mellom kameraene, som funksjon av avstand mellom objekt og kamera. Kameraavstand 500 m, utslag i feilvinkel 0,5°

Den andre testen gikk ut på å variere avstanden mellom de to kameraene og holde objektets posisjon konstant. Objektets avstand ble satt til 3 km og avstanden mellom kameraene varierte fra 100 m til 1 km. For hver avstand ble 1000 tilfeldige rotasjoner med utslag på 0,5° påført retningslinjen fra det ene kameraet. Resultatene kan ses i figur 4.4.



Figur 4.4: Feil i estimert posisjon som følge av relativ rotasjon mellom kameraene, som funksjon av avstand mellom kameraene. Objektavstand 3 km, utslag i feilvinkel 0,5°

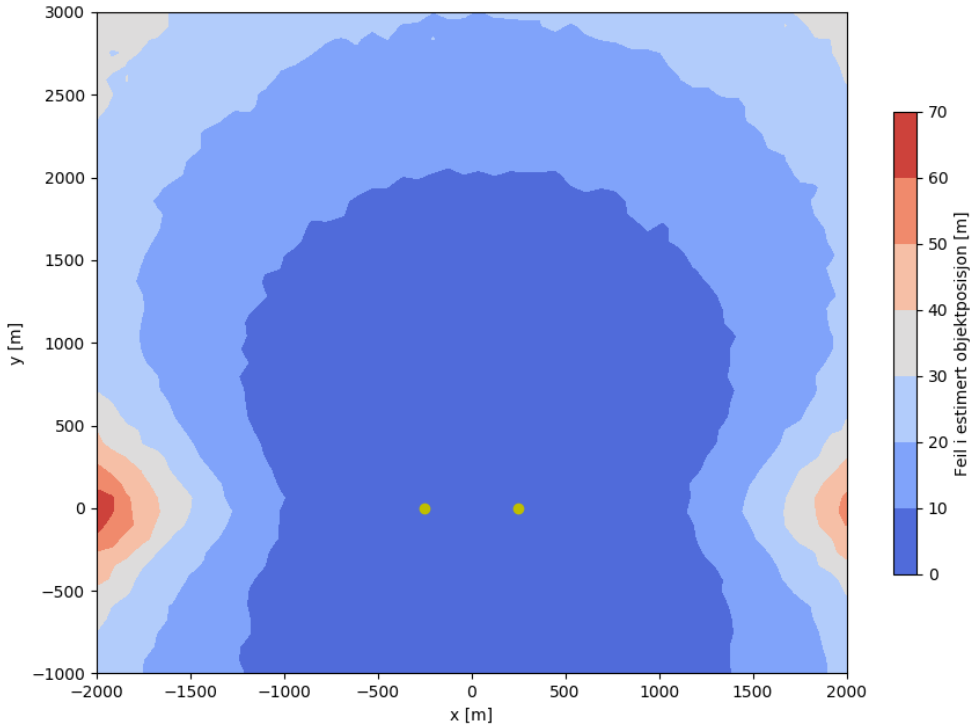
Den siste testen som ble gjennomført på rotasjonsavvik omhandler hvordan størrelsen på feilen i retningsvektoren påvirker estimatet av objektposisjonen. Også her ble objektets posisjon satt til 3 km og 1000 tester ble gjort for hvert vinkelutslag som ble testet. Kameraene ble satt til å ha 500 m mellomrom og vinkelutslaget varierte fra 0° til 1°. Resultatet er vist i figur 4.5.



Figur 4.5: Feil i estimert posisjon som følge av relativ rotasjon mellom kameraene som funksjon av utslaget til rotasjonen. Kameraavstand 500 m, objektavstand 3 km

4.1.3 Samlet Effekt av Avvik

Til sist skal den samlede effekten av disse avvikene undersøkes. For denne testen ble objektets x- og y-posisjon variert over et stort område på 4×4 km mens z-posisjonen holdes konstant på 300 m. For hver objektposisjon ble det gjort 200 målinger hvor støy ble introdusert. For hver måling ble det lagt på en tilfeldig rotasjon med utslag på 0,1° på den ene synslinjen, samt at de antatte kameraposisjonene ble forskjøvet fra de virkelige med 10 cm i en tilfeldig retning. For å vise hvordan objektets posisjon i forhold til kameraene påvirker estimeringsfeilen ble det evaluerte området valgt slik at det er stor variasjon i vinkelen mellom synslinjene ved objektet. Resultatene vises i figur 4.6. De gule prikkene angir posisjonen til de to kameraene som er plassert med 500 m avstand.



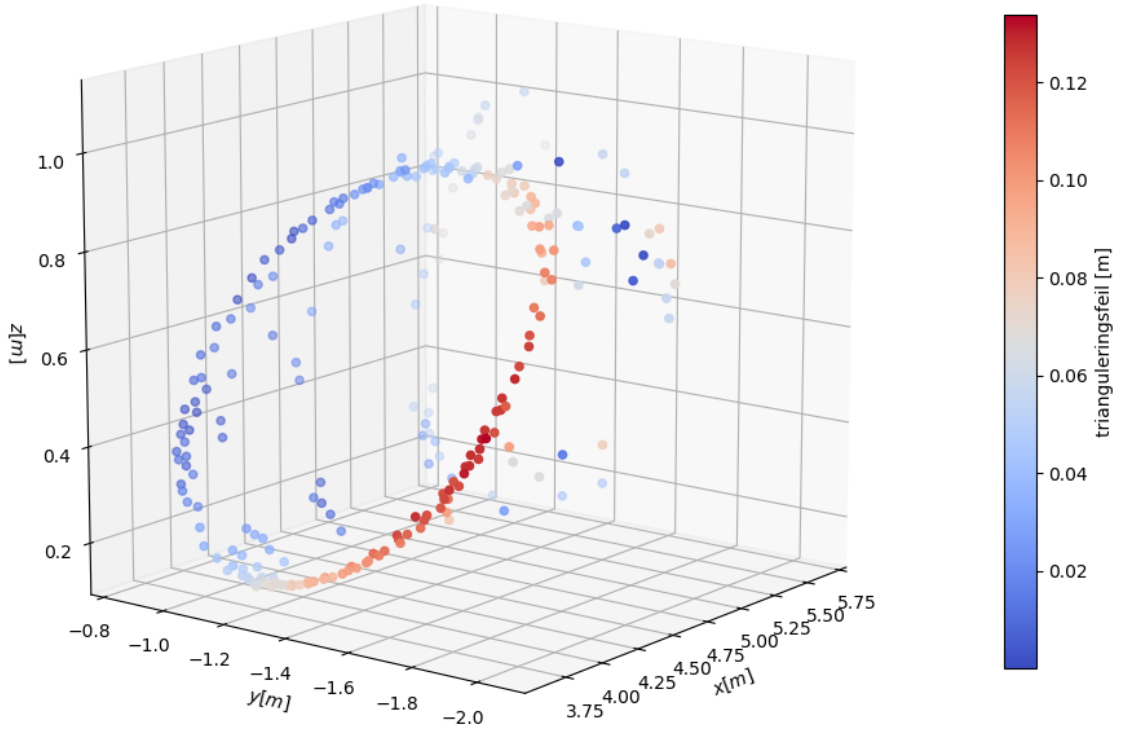
Figur 4.6: Feil i estimert posisjon som følge av flere typer avvik samlet, som funksjon av objektposisjon. Objekthøyde 300 m

4.2 Småskala Konsepttest

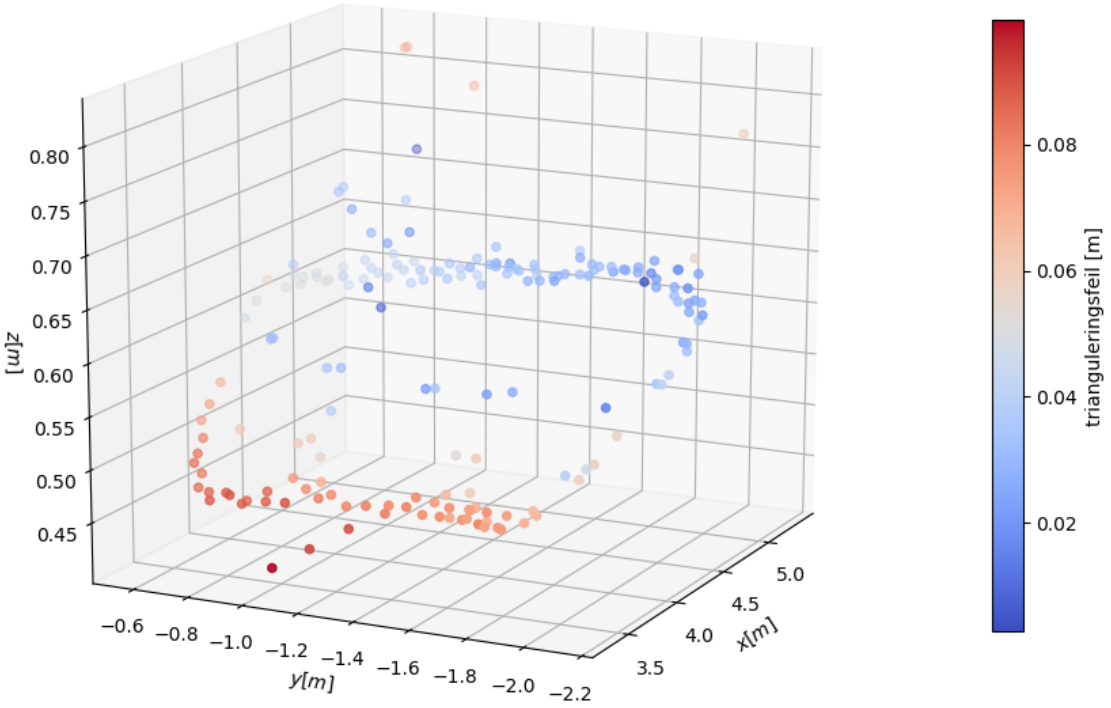
Mot sluttfasen av utviklingen av systemet ble det gjennomført en konsepttest av systemet. Målet med denne testen var i utgangspunktet bare som en bekreftelse på at konseptet i seg selv kunne fungere. Målet var ikke å produsere noen nøyaktige målinger. På dette tidspunktet var ikke data-maskinen som skulle fungere som vertsmaskin for systemet anskaffet enda. Testen ble derfor gjennomført med en annen maskin som hadde viktige mangler. Denne maskinen hadde ikke et nettverkskort som støttet 10GigE og Oryx-kameraet måtte derfor overføre bilder over vanlig 1 Gbit ethernet istedet. Maskinen hadde heller ikke et dedikert skjermkort som gjorde de-bayering av bildene og subsekvent konvertering til OpenCV format relativt tregt. Alt dette resulterte i at systemet kun oppdaterte seg i en omtrentlig hastighet på 1 Hz, eller én gang i sekundet. Minst 10Hz skulle vert ønskelig, men dette holdt til denne konsepttesten.

Testen ble gjennomført ved at en svart firkant, på ca. 3×3 cm, ble projisert på et lerret omtrent 4,5 m foran kameraene og litt til siden. Kameraene ble plassert på et bord og hadde omtrent 1 m mellom seg. Disse posisjonene ble kun målt for hånd og kameraenes rotasjon ble kun rettet opp på øyemål. Nøyaktighet var som sagt ikke målet for denne testen. Det ble utført to typer tester, en hvor firkanten beveget seg frem og tilbake langs en horisontal linje på lerrettet, og en hvor firkanten beveget seg i en ellipse, mot klokken.

I figur 4.7 og 4.8 er de målte posisjonene fra denne testen plottet. Fargen på punktene angir hva som var den miste avstanden mellom de to linjene som ble triangulert for å finne det tilsvarende punktet. Enhetene er i m.



Figur 4.7: Resultat av konsepttest med følging i ellipse



Figur 4.8: Resultat av konsepttest med følging i rett linje

4.3 Test med drone

Ved utgangen av prosjektet ble det lagt vekt på å få testet systemet i mer virkelighetsnære omgivelser enn den testen som ble beskrevet i kapittel 4.2. Det beste som kunne blitt utført av en slik test ville vært å bruke systemet til å måle posisjonen til et helikopter, eller fly, slik systemet egentlig er beregnet for. Dette var desverre ikke mulig å få til innenfor de tidsrammene som foreligger på en oppgave som dette. Valget falt derfor på å heller teste systemet med en drone som mål. Dette er hensiktsmessig da det er enkelt å få dronen til å fly akkurat som ønsket.

Systemet ble satt opp på Brattåsen skytebane. Dette stedet ble valgt fordi det er et relativt stort åpent område med lite bebyggelse og svært lite trafikk rundt. Dette er viktige sikkerhetshensyn da dronekollisjoner ikke er umulig. Testen ble utført på dagtid, i godt lys, og været var delvis overskyet. Kameraene ble satt opp med 11 m avstand mellom seg.

Før testen begynte ble posisjonen til begge kameraene målt med differensiell GPS, det samme ble to synlige punkter i terrenget. Denne informasjonen ble brukt til å bestemme kameraenes orientering og posisjon i forhold til ENU systemet via fremgangsmåten som er lagt frem i kapittel 3.5.

Den differensielle GPSen ble så montert på dronen slik at det foreligger en svært nøyaktig måling av dronens posisjon som kan sammenlignes med posisjonen målt av kamerasystemet. I beste tilfelle har en differensiell GPS en nøyaktighet på omlag 1 cm. Dette avhenger av hvor mange GPS-satellitter den og den tilhørende basestasjonen får kontakt med. Skytebanen der testen foregikk er omringet av forholdsvis høye trær, dette begrenset tilgangen til satellitter noe. Derfor er nøyaktigheten på DGPS-målingene omlag 10 cm for størsteparten av denne testen.

I kapittel 4.2 ble oppdateringshastigheten til systemet nevnt som en utfordring. Under den testen var maskinen som styrte systemet ikke utstyrt hverken med et dedikert skjermkort eller et 10GigE-kompatibelt nettverkskort. Under denne testen var begge disse komponentene installert i maskinen. Oppdateringsfrekvensen har derfor økt til litt over 2 Hz for denne testen. Dette er fortsatt ikke fullt så høyt som er ønskelig. Mulige måter å forbedre dette på diskuteres i kapittel 6.

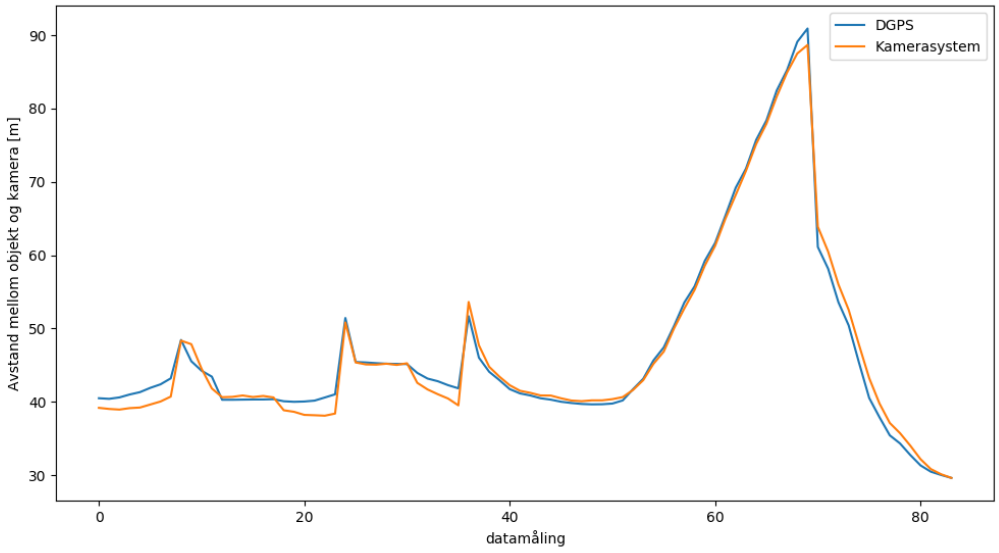
Grunnet noen tekniske problemer med kamerasystemet samt med datamaskinen det hele kjørte på, kombinert med dronenes begresende batterikapasitet, ble det ikke samlet inn fullt så mye data som hadde vært ønskelig. Det foreligger likevel nok resultater til å kunne si noe om systemets funksjon og nøyaktighet. Disse resultatene vil presenteres her.

Det første plotet som legges frem viser avstanden mellom dronen og det ene kameraet i systemet, som målt både av kamerasystemet og av DGPSen montert på dronen, over serien med målinger. Merk at plutselige hopp i avstand skyldes starten på en ny måleserie, ikke at dronen faktisk beveget seg så hurtig. Se figur 4.9.

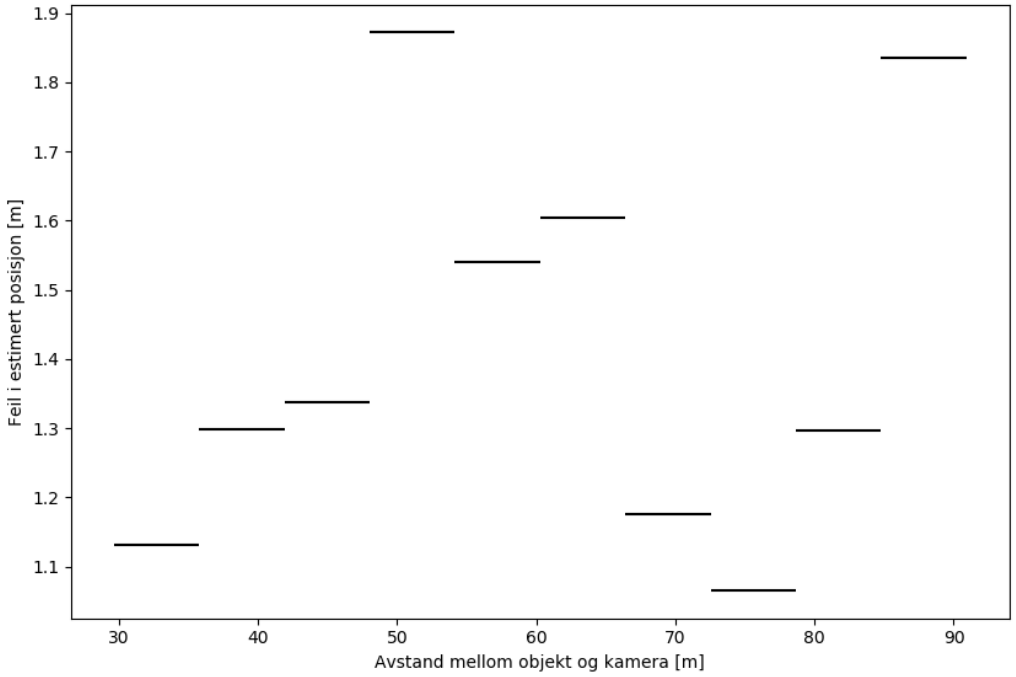
Det neste plotet viser hvordan normen av avviket mellom dronens posisjon som målt av kamerasystemet og DGPSen utvikler seg med avstand mellom dronen og kamerasystemet. Dataen er kvantisert basert på avstanden og gjennomsnittet av de målingene som faller innenfor samme kvantil er vist. Se figur 4.10.

Figur 4.11 viser hvordan dronens hastighet påvirker kamerasystemets posisjoneringsfeil og figur 4.12 viser det samme men her er dronens hastighet delt på avstanden mellom drone og kamera. Figur 4.12 gir altså et inntrykk av hvordan posisjoneringsfeilen påvirkes av dronens hastighet slik den synes for kameraet.

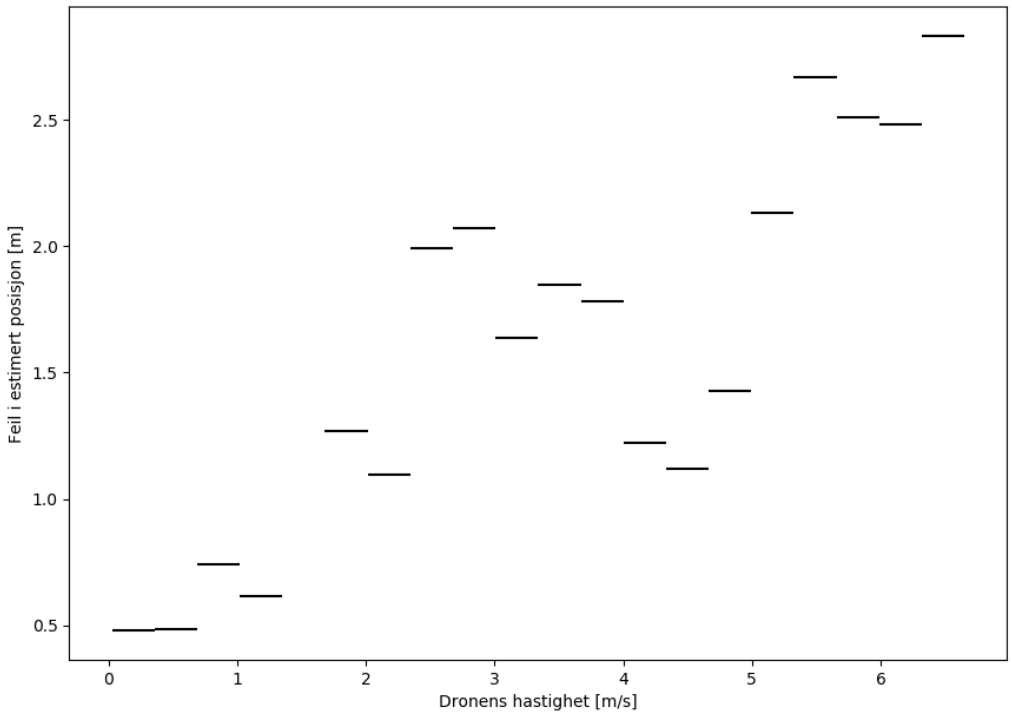
Det siste plottet viser et histogram over alle de målte avvikene. Se figur 4.13.



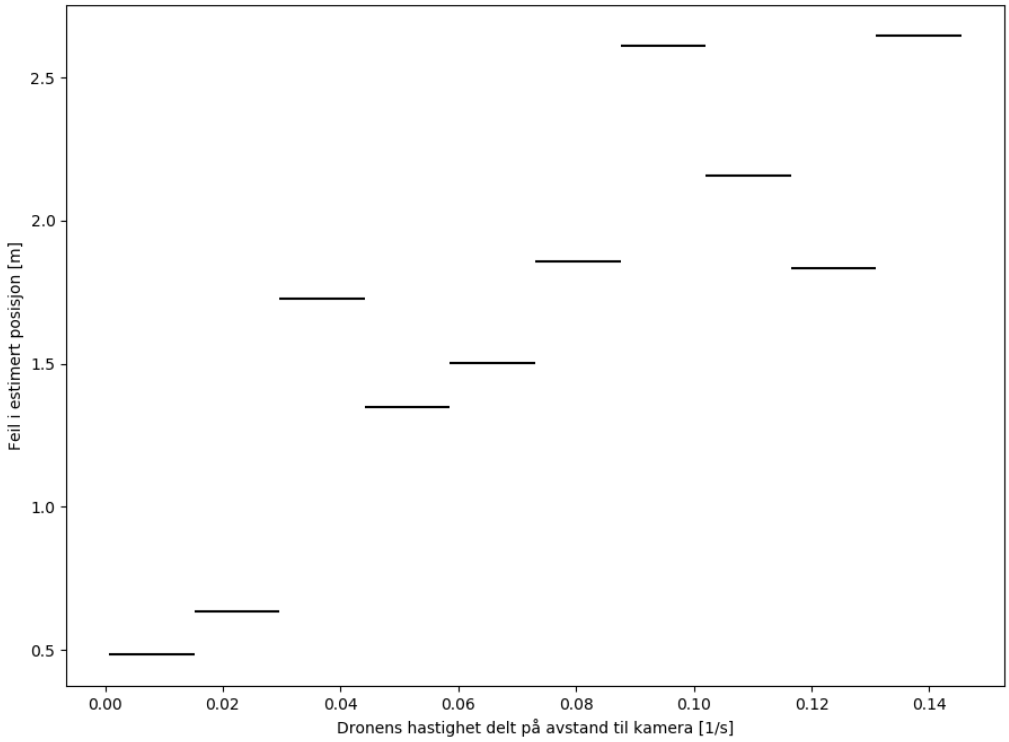
Figur 4.9: Avstand mellom objekt og kamera målt av kamerasystem og DGPS



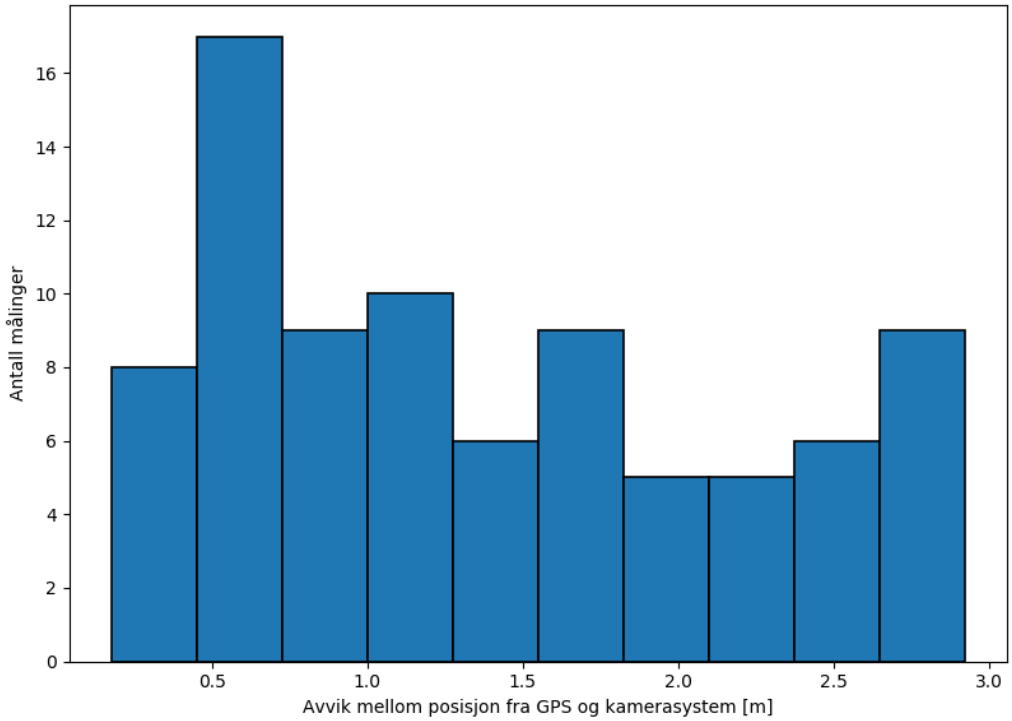
Figur 4.10: Avvik mellom posisjon målt av kamerasystem og DGPS, som funksjon av avstand mellom kamera og objekt



Figur 4.11: Kamerasystemets posisjoneringsfeil som funksjon av dronens hastighet



Figur 4.12: Kamerasystemets posisjoneringsfeil som funksjon av dronens hastighet delt på avstand mellom kamera og drone



Figur 4.13: Histogram over størrelsen på posisjoneringsfeil

Kapittel 5

Diskusjon

I foregående kapittel ble resultatene av de forskjellige delene av arbeidet med prosjektet lagt frem. Dette kapittelet er viet til tolkning og diskusjon av disse resultatene som vil skape grunnlaget for konklusjonen.

5.1 Simulering

Resultatene fra simuleringen viser hvor stor påvirkningskraft de forskjellige feilkildene i systemet har. Figur 4.1 og 4.2 viser at avviket som følge av feil i estimert kameraposisjon ikke er veldig stor. I disse simuleringene er det brukt et avvik på 50 cm. Dette er et stort avvik og med godt utstyr burde det være enkelt å holde dette avviket under 10 cm i praksis. Det er også tydelig av feilen blir enda mindre ettersom avstanden mellom kameraene øker. I figur 4.1 er det brukt en avstand på 50 m mellom kameraene, noe som er halvparten av den minste avstanden som tenkes brukt i praksis. I simuleringen med 500 m mellom kameraene er feilmarginen svært liten, selv med et avvik på 50 cm, og når såvidt 3 m på 5 km avstand til objektet. Det som også er verdt å merke her er at feilen ser ut til å stige tilnærmet lineært med avstanden til objektet. Alt i alt betyr dette at denne feilkilden ikke er den viktigste å kontrollere for å oppnå nøyaktighet i systemet.

Rotasjonsavviket som vises i figur 4.3 forteller en annen historie. Her er det brukt utslag på en halv grad som ikke er særlig mye, og er en feil som lett kan oppstå som følge av unøyaktighet i kalibrering. Likevel fører dette til veldig store avvik, med en gjennomsnittlig posisjonsfeil på 25 m på en avstand under 1,5 km. Dette er langt utenfor det ønskelige feilmargins-området for systemet. I simuleringen som genererte 4.3 var kameraene plassert 500 m fra hverandre. Fra resultatene vist i figur 4.4 er det tydelig at en større kameraavstand er ønskelig da det drastisk

minker effekten av et slikt rotasjonsavvik.

Det er også tydelig fra figur 4.3 at feilen ser ut til å vokse raskere enn lineært med avstanden mellom objekt og kamera. Med tanke på det som ble presentert i kapittel 2.1 er det rimelig å anta at denne raske veksten skyldes at feilen vokser med kvadratet av sinus av vinkelen som spennes mellom de to kameraene fra objektet.

Det er dette som gjør at feilen fort vokser seg veldig stor når objektet beveger seg lengre unna. Dette er altså er feilkilde som det er veldig viktig å kontrollere for når nøyaktigheten til systemets målinger skal ivaretas. Det er derfor nødvendig med en metode for å sikre at den relative rotasjonen til kameraene er kjent med svært høy nøyaktighet.

Når det kommer til størrelsen på feilen i rotasjon, som vist i figur 4.5, vokser feilen som følge av denne tydelig lineært. Dette er gode nyheter da dette vil si at det alltid er mye å vinne på å redusere denne feilkilden. Likevel viser figuren også at stigningen til denne lineære sammenhengen er svert bratt for dette systemet. Det er ingen tvil om at systemets nøyaktighet er svært sårbart for slike rotasjonsfeil og det viktigste punktet for å sikre systemets pålitelighet er å minimere denne typen feil.

Til sist må figur 4.6 diskuteres. Denne figuren viser det gjennomsnittlige posisjonsavviket for objektposisjoner rundt kameraene. Det første som er verdt å merke seg ved denne figuren er hvordan feilen stiger mye raskere langs linjen som dannes av de to kameraene. Dette er fordi vinkelen mellom de to sysnlinjene er tilnærmet parallelle når objektet befinner seg langs denne linjen. Dette viser viktigheten av å sette opp systemet på en slik måte at objektet ikke befinner seg på linje med kameraene, så vidt det er mulig.

Bortsett fra akkurat disse punktene rundt $y = 0$ virker det som det nesten utelukkende er avstand fra kameraene som bestemmer feilen. I denne simuleringen er det brukt 0,1° utslag i rotasjonsavvik og 10 cm i utslag på posisjonsavvik. Likevel ser vi at feilen når 10 m allerede på en avstand på 2 km. Dette er mer enn det som er ønskelig og viser hvor viktig det er å kontrollere for disse feilkildene for å få systemet til å produsere nøyaktige målinger.

5.2 Konsepttest

For å vise at prosjektets underliggende konsept kan fungere og for å verifisere funksjonen til de ulike delene av systemet ble det gjennomført en konsepttest. Utførelsen av testen er beskrevet i kapittel 4.1.2. De to sentrale resultatene som vil diskuteres her er figur 4.7 og 4.8.

Det første som er tydelig ved å se på disse resultatene er at de helt klart ligner på de formene de skulle representere. Punktene i 4.7 følger tydelig formen til en ellipse og punktene i 4.8 ligger

klart på linje. Dette viser at konseptet bak prosjektet, å bruke to kamera og triangulere deres sysnlinjer mot et objekt, kan fungere i praksis. Likevel er det noen åpenbare problemer i disse resultatene. Som beskrevet i kapittel 4.1.2 ble målet i denne testen projisert på et lerret. Dette lerretet hang rett ned og det derfor burde alle de målte punktene havne i et plan som er omtrent parallelt med z-aksen. Dette er åpenbart ikke tilfellet for disse resultatene.

I figur 4.7 virker det som at ellipsen ligger i et plan hvis normal står nærmere 45° på z-aksen, enn 90° slik det burde. I 4.8 er punktene tydelig delt i to, en linje med en høyere x-, og z-verdi, og en linje med lavere verdier på disse to aksene. Det kan se ut til at disse linjene ligger i omtrent samme planet som punktene i 4.7. Da systemet kun ble kalibrert for hånd ville det ikke være overraskende om kameraene ikke var helt perfekt justert i samsvar med retningen til lerretet. Likevel er denne vinkelen langt større enn det som kan forventes å skyldes feil i oppsettet.

Videre er det også en klar sammenheng mellom trianguleringsfeilen tilhørende hvert punkt og hvilken av de to linjene punktet tilhører. I 4.7 finnes en lignende sammenheng hvor punktene med størst negativ y-verdi har klart høyere trianguleringsfeil enn de med y-verdi nærmere null.

Det kan være flere grunner til disse åpenbare sammenhengene mellom punktenes posisjon og deres trianguleringsfeil. I første omgang ble det foreslått at sammenhengen i 4.7 stammer naturlig fra objektets posisjon. Aksene angir målets posisjon relativt til 360° kameraet. Dermed er det altså punktene som er lengst unna kameraet som har merkbart høyest trianglulerings feil. Dette er ikke unaturlig, og denne sammenhengen er tydelig å se fra resultatene lagt frem i kapittel 4.1. Denne tolkningen holder likevel ikke vann når figur 4.8 betraktes.

I denne testen beveget målet seg fra side til side. Med tanken om at det er målets avstand til kameraene som dikterer trianguleringsfeilen forventes det da at trianguleringsfeilen følger en gradient langs linjen. En gradient hvor feilen er liten når målet er på delen av linjen nermest kameraene, og har en liten negativ y-verdi, og som vokser og blir størst når målet er lengst unna kameraene, og har en større negativ y-verdi. Dette er ikke det som synes i figur 4.8. Her synes feilen å være tilnærmet lik langs hele linjen, fra høyre til venstre. Forskjellen ligger i om punktet tilhører den øvre, bakre gruppen, eller den fremre, lavere. Det er også tydelig at den fremre gruppen har en gjennomsnittlig høyere feil en den bakre, selv om disse punktene ble estimert til å være nærmere kameraene. En annen forklaring må ligge bak.

En annen forklaring på disse feilene kan ligge i oppdateringsfrekvensen på Oryx-kameraet under denne testen. Som nevnt i kapittel 4.1.2 ble denne testen utført før alt av maskinvare var på plass. Dette medførte en svært lav oppdateringsfrekvens på systemet og særlig på Oryx-kameraet. Dette gjorde at bildene fra dette kameraet lå litt bak objektets faktiske posisjon. Dette gjorde slik at synslinjen fra Oryxen mot objektet pekte mot et punkt objektet hadde vært tidligere. Denne posisjonen samsvarte da ikke med posisjonen målt fra Ladybugen som var mer oppdatert.

Dette gjør at det ikke bare er objektets posisjon som påvirker målingene fra systemet, men også objektets retning og hastighet. Når objektet beveget seg mot høyre vil Oryxens synslinje peke mot en tidligere posisjon av objektet som da blir noe til venstre for den faktiske posisjonen. Under testen var Oryxen plassert til høyre for Ladybugen, altså i negativ y-retning. Når Oryxens synslinje peker mot et punkt for langt til venstre og er plassert til høyre for Ladybugen fører dette til at de to synslinjene møtes på et punkt nermere enn det som er objektets reelle posisjon. Likedan hvis objektet er på vei mot venstre vil synslinjen fra Oryxen peke for langt mot høyre og de to synslinjene vil møtes ved et punkt lengre bak en det som er reellt.

Dette forklarer hvorfor punktene i 4.8 er delt i to grupper, en nærmere kameraene og en lenger unna. Den ene gruppen, den nermest kameraene, er målinger gjort mens objektet beveget seg mot høyre og den andre er målinger som er gjort mens objektet beveger seg mot venstre.

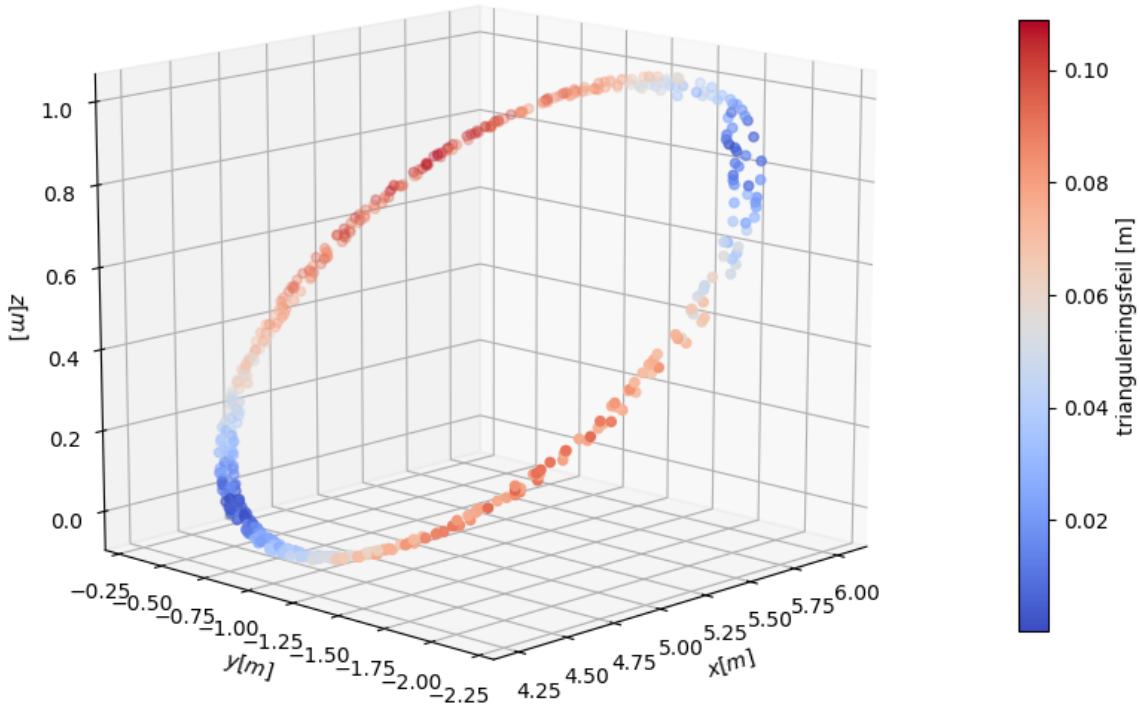
For å undersøke om dette kunne være bakgrunnen for de observerte feilene ble det gjennomført en simulering. Simulatoren som ble brukt er den samme som er diskutert i kapittel 3.2 og som produserte resultatene lagt frem i kapittel 4.1. I denne simuleringen ble konsepttesten forsøkt gjenskapt, både i form av målets posisjon og bevegelse, samt kameraenes relative posisjon. Linjen fra kamera 2, som representerer Oryxen, ble satt til å peke på objektets posisjon noe bakover i tid sammenlignet med linjen fra kamera 1, som representerer Ladybugen. Resultatene fra denne simuleringen er vist i figur 5.1 og 5.2.

Disse figurene viser tydelig den samme forskyvelsen av ellipsen og oppdelingen av linjen slik som i figur 4.7 og 4.8. Det er derfor god grunn til å tro at det var denne feilen i systemet som ga opphav til feilestimeringen av posisjonene. Det som derimot er forskjellig mellom resultatene fra konsepttesten og simuleringen er fordelingen av trianguleringsfeilen. I figur 4.7 er denne feilen tydelig fordelt slik at den er stor til høyre og liten til venstre. I 5.1 er det ganske likt at feilen er større til høyre, men dette speiler seg også på venstre side. Her er det istedet toppen og bunnen av ellipsen som har lavest trianguleringsfeil.

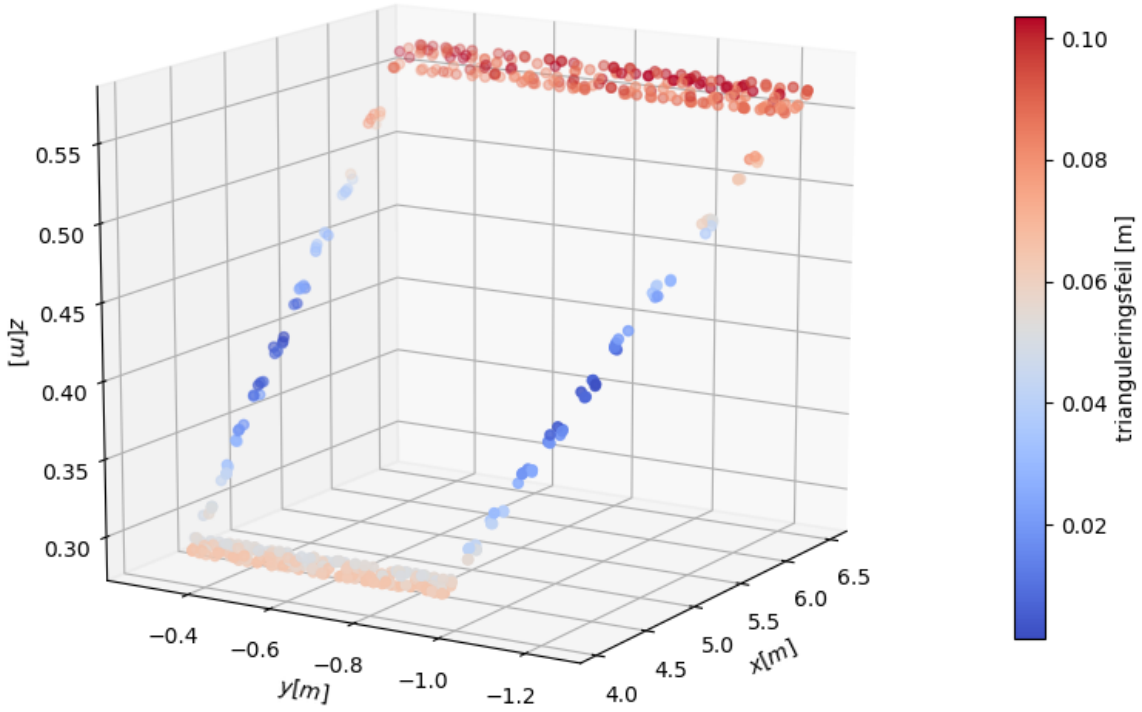
Den samme type avvik kan sees mellom figur 4.8 og 5.2. Under konsepttesten var det punktene som ble evaluert til å være nermest kameraene som hadde høyest feil, mens de som ble plassert lengre unna hadde en markant mindre feil. Under simuleringen virker det å ha vært motsatt, da punktene som ble plassert lengre unna kameraene virker å ha en noe større feil enn de som ble plassert nermere.

Denne forskjellen mellom simuleringene og resultatene fra konsepttesten antas å komme fra unøyaktigheter i oppsettet av konsepttesten som ikke kan gjenskapes nøyaktig i en simulering.

Denne konsepttesten viser at den grunnleggende virkemåten til systemet er gjennomførbar i praksis. Testen har også belyst hvorfor nøyaktig tidsstempling av informasjon er svært viktig for et system som dette som belager seg på to uavhengig produserte målinger.



Figur 5.1: Resultat av simulert konsepttest hvor målet beveger seg i en ellipse



Figur 5.2: Resultat av simulert konsepttest hvor målet beveger seg langs en rett linje

5.3 Test med Drone

I kapittel 4.3 ble det beskrevet hvordan det ble gått frem for å teste systemet i mer virkelighetsnære omgivelser. Målet for denne testen var å produsere resultater som kunne si noe om systemets funksjonalitet og nøyaktighet. Disse resultatene ble lagt frem via figurene 4.9 til 4.13. I dette kapittelet vil det diskuteres hva disse resultatene viser og hva dette betyr for systemet, dets funksjonalitet og dets nøyaktighet.

Først vil figur 4.9 diskuteres. Det første som kommer frem fra denne figuren er at de to linjene, som viser avstand mellom drone og kamera som målt både via DGPS og av kamerasystemet, tydelig følger hverandre. Fra dette kan det konkluderes at systemet, i alle fall til en hvis grad, klarer å følge dronen mens den flyr og produserer posisjonsestimat som er rimelige.

På den annen side viser figuren også store avvik flere steder. Disse avvikene har ingen tydelig sammenheng med avstanden mellom dronen og kameraet og ser ut til å oppstå omtrent tilfeldig. Dette tyder på at opphavet til denne feilen ikke ligger i en konstant feil, slik som feil i antatt relativ posisjon eller rotasjon av kameraene. En slik feil ville tydelig vokst med avstanden mellom drone og kamera slik som vises i kapittel 4.1.

Figur 4.10 viser mer direkte hvordan posisjoneringsfeilen avhenger av avstanden mellom drone og kamera. Også fra denne figuren er det tydelig at det er svært liten sammenheng mellom posisjoneringsfeilen og avstanden. Denne figuren sier også noe om størrelsen på posisjonsfeilen hvor gjennomsnittet for hvert kvantil i avstand her ligger mellom 1 og 2 meter. Dette er en feil som er betydelig større en det som er ønskelig, særlig med tanke på at avstandene mellom drone og kamera i denne testen er under 100 m.

I figur 4.11 kommer en klarere sammenheng frem. Fra denne figuren virker det tydelig at dronens hastighet under testen hadde en innvirkning på kamerasystemets evne til å posisjonere dronen nøyaktig. Selv om trenden ser ut til å gå nedover for de mellomstore hastighetene fra 2,5 m/s til 5 m/s er trenden tydelig positiv når hele datasettet ses under ett.

Denne trenden kommer enda tydeligere frem når vi justerer dronens hastighet med avstanden mellom den og kamera. Figur 4.12 viser denne sammenhengen. Her har vi ikke den samme negative trenden midt i datasettet og ser tydelig at høyere hastighet relativt til avstand gir en betydelig høyere posisjoneringsfeil.

Spørsmålet er da hva slags feil kan oppstå som øker med hastigheten til objektet som skal posisjoneres. I diskusjonen rundt resultatene fra konsepttesten, som finnes i kapittel 4.1.2, trekkes oppdateringsfrekvensen av systemet ut som en svært sannsynlig feilkilde i systemet. Da denne testen ble gjennomført var oppdateringshastigheten økt fra omlag 1 Hz til litt over 2 Hz. Dette er en betydelig økning, men det er fortsatt ønskelig at systemet skal ha en oppdateringsfrekvens på over 10 Hz. Det er derfor sannsynlig at dette kan være en feilkilde.

Når systemet beregner linjen som peker fra Oryx-kameraet mot objektet antas det at objektet er sentrert i bildet. Denne antagelsen skal holdes sann av regulatoren som styrer kameraet på en slik måte at objektet det følger hele tiden sentreres i bildet. Når oppdateringsfrekvensen på denne delen av systemet er lav vil ikke regulatoren klare denne oppgaven perfekt, særlig når objektets hastighet er stor som sett fra kameraet.

Denne feilkilden kan heller ikke være den eneste feilen som påvirker systemet. I figur 4.11 og 4.12 er det tydelig at systemet har en betydelig feil også når dronen står tilnærmet stille. Når dronen står stille skulle Oryx-kameraet klare å sentrere objektet selv med lav oppdateringsfrekvens. Likevel har systemet en feil med gjennomsnitt på omlag 0,5 m selv når dronens hastighet er tilnærmet null.

Figur 4.13 gir et bilde på hvor stor feil som kan forventes av systemet. Her vises det at den største delen av posisjoneringsfeilen ligger i den lavere delen av spekteret og feil på rundt 0,5 m utgjør den største gruppen. Likevel kan feilen bli svært stor og i noen tilfeller nærme seg 3 meter.

Det er rimelig å anta at det finnes minst en annen feilkilde i systemet i tillegg til feilen som oppstår på bakgrunn av den lave oppdateringsfrekvensen. Dette kommer frem av at systemet posisjonerer med betydelig feil også når målet står stille. Det er derimot vanskelig å si hva denne feilkilden kan være da feilen på bakgrunn av oppdateringsfrekvensen er så stor og har mye tilfeldig varians. Denne feilen overskygger i stor grad de andre feilene i systemet og gjør dem vanskelige å oppdage.

Da posisjonen til kameraene for denne testen ble målt med differensiell GPS er deres relative posisjon kjent med svært høy nøyaktighet. I kapittel 4.1 ble det også vist at denne typen feil ikke har veldig stor innvirkningskraft på systemets nøyaktighet, særlig ikke på små avstander som det var under denne testen. Det er derfor mer rimelig å tro at denne andre feilkliden kan komme fra at kameraenes orientering i forhold til ENU systemet ikke er kjent med høy nok presisjon.

Før systemet ble tatt i bruk og målinger ble gjort ble fremgangsmåten lagt frem i kapittel 3.5.2 fulgt for å fastslå kameraenes orientering i forhold til ENU systemet. Likevel er det vanskelig å velge ut den pikselen som aller best representerer punktet som ble målt. Det blir også kun brukt to punkter til å bestemme orienteringen av kameraene. Ved å måle opp flere punkter og beregne den orienteringen som passer best med alle punktene vil det være mulig å oppnå et mer nøyaktig estimat av kameraenes orientering.

Alt i alt viser denne testen at dette systemet har betydelige utfordringer. De posisjoneringsfeilene som er gjort under denne testen er betydelig større en det som er ønskelig for dette systemet om det skulle tas i reelt bruk. Likevel har testen også gitt noe innsyn i hvilke feil som er skyld i de posisjoneringsfeilene som er observert. Dette gir en klar retning for videre arbeid. Videre har testen også vist at systemet, selv med de utfordringene som er diskutert her, har et konsept som kan fungere i praksis.

Kapittel 6

Konklusjon og Videre Arbeid

Denne oppgaven har lagt frem det arbeidet som er blitt gjort med å utvikle og utprøve et kamera-basert sensorsystem for posisjonering av fly. Det er gitt en grundig gjennomgang av systemets oppbygging både i form av fysiske komponenter og maskinvare. Det er også lagt frem hvordan utprøvelsene av systemet er blitt gjennomført og resultatene av disse.

Disse testene har gitt viktig innsikt i systemet som helhet og utpekt hvilke deler av systemet som fungerer godt, og hvor systemet slik det står i dag har utfordringer. Den delen av systemet som har fungert best i disse testene har helt klart vært prosessen knyttet til 360°-kameraet. Prosessen her, med forgrunnsdeteksjon og målfølging med kalman filter, har fungert ypperlig. Denne delen av systemet har alltid funnet og fulgt det ønskede målet. Da det har vært store unøyaktigheter i andre deler av systemet har det ikke vært mulig å bedømme nøyaktigheten av den data som denne delen av systemet har produsert.

Den delen av systemet som har stått for de store feilene og unøyaktighetene er i stor grad systemet knyttet til Oryx-kameraet. Slik systemet er i dag er ikke metoden som brukes for å produsere retningen fra Oryxen til objektet god nok. Systemet klarer ikke å holde objektet i sentrum av bildene fra Oryxen, noe som er en antagelse for systemets operasjon. Dette fører til at de linjene som skal peke fra Oryxen til objektet har store unøyaktigheter som videre gir store feil i posisjoneringen av objektet. Om systemet skal tas i bruk under faktiske forhold må dette utbedres.

Likevel er det klart at den grunnleggende tanken bak systemet er gjennomførbar i praksis. Testene har vist at systemet klarer å følge objektene på himmelen og klarer å posisjonere dem, bare med svært lav presisjon slik det står nå. Simuleringene som er gjort viser også at dette systemet kan oppnå høy presisjon, også på lange avstander, men at dette krever svært høy nøyaktighet i alle delene av systemet.

Systemet må altså utbedres før det kan tas i bruk i virkelige forsøk. Denne oppgaven har i hovedsak pekt på oppdateringsfrekvensen til Oryx-kameraet som det største hinder for systemets presisjon. Dette er derfor et logisk sted å begynne å se på mulige forbedringer. Det er to mulige løsninger på dette problemet. Den ene er å øke oppdateringsfrekvensen til denne delen av systemet, den andre er å ikke bruke antagelsen om at objektet som følges befinner seg i sentrum av bildet og dermed langs kameraets optiske akse.

Å øke systemets oppdateringsfrekvens er uansett en forbedring som er ønskelig. Med flere målinger per sekund vil det være mulig å se på flere målinger sammen for å forbedre posisjonsestimatet. I tillegg vil det altså hjelpe betydelig på nøyaktigheten fra Oryxen. I dette prosjektet er det brukt en kraftig datamaskin med toppmoderne maskinvare som sentralsystem. Likevel har oppdateringsfrekvensen vært svert lav. Som nevnt i kapittel 3.4.5 ble det forsøkt å øke utnyttelsen av maskinens ressurser ved å dele opp programmet i flere tråder. Dette lyktes ikke grunnet at de underliggende grensesnittene ikke kunne kjøres samtidig uten å hindre hverandre.

En mulig løsning ville derfor være å dele systemet i to, slik at hvert kamera er tilknyttet hver sin datamaskin. I tester gjort under utviklingen av dette systemet er det oppnådd en oppdateringsfrekvens på over 15 Hz på hver enkelt del av systemet når det kjører alene. Dette ville vært en stor økning fra det nåværende 2 Hz. Dette ville selvsagt også krevd at de to datamaskinene kunne kommunisere. Dette kan gjøres enkelt via ethernet da det kun er små mengder data, i form av to vektorer, som skal overføres.

En økning i systemets oppdateringsfrekvens vil kunne forbedre nøyaktigheten til systemet betraktelig i tillegg til å la systemet produsere målinger raskere. Likevel er det ikke garantert at systemet alltid vil klare å holde det fulgte objektet presist i sentrum av bildene fra Oryxkameraet. Særlig hvis objektet som skal følges manøvrerer kraftig eller beveger seg svært fort. Det er derfor ønskelig å bli kvitt denne antagelsen. Dette kan gjøres ved å estimere de interne parameterene til kameraet. Dette vil gjøre det mulig å beregne en linje som tilsvarer et vilkårlig piksel i bildet, ikke bare det midterste. Det finnes fritt tilgjengelig programvare som gjør det mulig å estimere disse parameterene.

Begge disse forslagene er utførbare ting som kan forbedre presisjonen til systemet betraktelig. En annen måte å gå frem vil være å fokusere mer på den delen av systemet som allerede fungerer svært godt, nemlig systemet rundt Ladybugen. Ved å utstyre systemet med et nytt Ladybugkamera vil det produseres to linjer mot målet med god nøyaktighet. Dette vil også gjøre at Oryxkameraet ikke lenger trenger å søke for å finne målet, men kan rettes mot det direkte. Et system som består av to Ladybug-kamera i tillegg til Oryx-kameraet vil da også kunne produsere tre linjer mot målet. Dette gjør at det kan trianguleres tre forskjellige posisjoner for hvert tidssteg og gjennomsnittet av disse målingene kan brukes for å oppnå enda høyere presisjon.

Bibliografi

[1] Gary Bishop, Greg Welch mfl. «An introduction to the Kalman filter». I: *Proc of SIGGRAPH, Course* 8.27599-3175 (2001), s. 59.

[2] Samuel Blackman og Robert Popoli. «Design and Analysis of Modern Tracking Systems». I: *Artech house* (1999).

[3] Thierry Bouwmans, Fida El Baf og Bertrand Vachon. «Background modeling using mixture of gaussians for foreground detection-a survey». I: *Recent Patents on Computer Science* 1.3 (2008), s. 219–237.

[4] FLIR Integrated Imaging Solutions Inc. *FLIR PTU-5*. Datasheet. Versjon 1.0. Apr. 2017. URL: https://www.flir.com/globalassets/imported-assets/document/17-0891-oem-cor-ptu-5-datasheet_final_v2-web.pdf.

[5] FLIR Integrated Imaging Solutions Inc. *Ladybug 5+ Technical Reference*. Technical Reference. Versjon 1.0. Sep. 2017. URL: https://www.ptgrey.com/support/downloads/10863.

[6] FLIR Integrated Imaging Solutions Inc. *ORX-10G-123S6 Technical Reference*. Technical Reference. Versjon 1.0. Okt. 2017. URL: https://www.ptgrey.com/support/downloads/10904.

[7] Chris Stauffer og W Eric L Grimson. «Adaptive background mixture models for real-time tracking». I: *cvpr*. IEEE. 1999, s. 2246.

[8] Zoran Zivkovic. «Improved adaptive Gaussian mixture model for background subtraction». I: *Proceedings of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004.* Bd. 2. Aug. 2004, 28–31 Vol.2. DOI: 10.1109/ICPR.2004.1333992.

[9] Zoran Zivkovic og F Van der Heijden. «Efficient adaptive density estimation per image pixel for the task of background subtraction». I: *Pattern Recognition Letters* 27 (mai 2006), s. 773–780. DOI: 10.1016/j.patrec.2005.11.005.

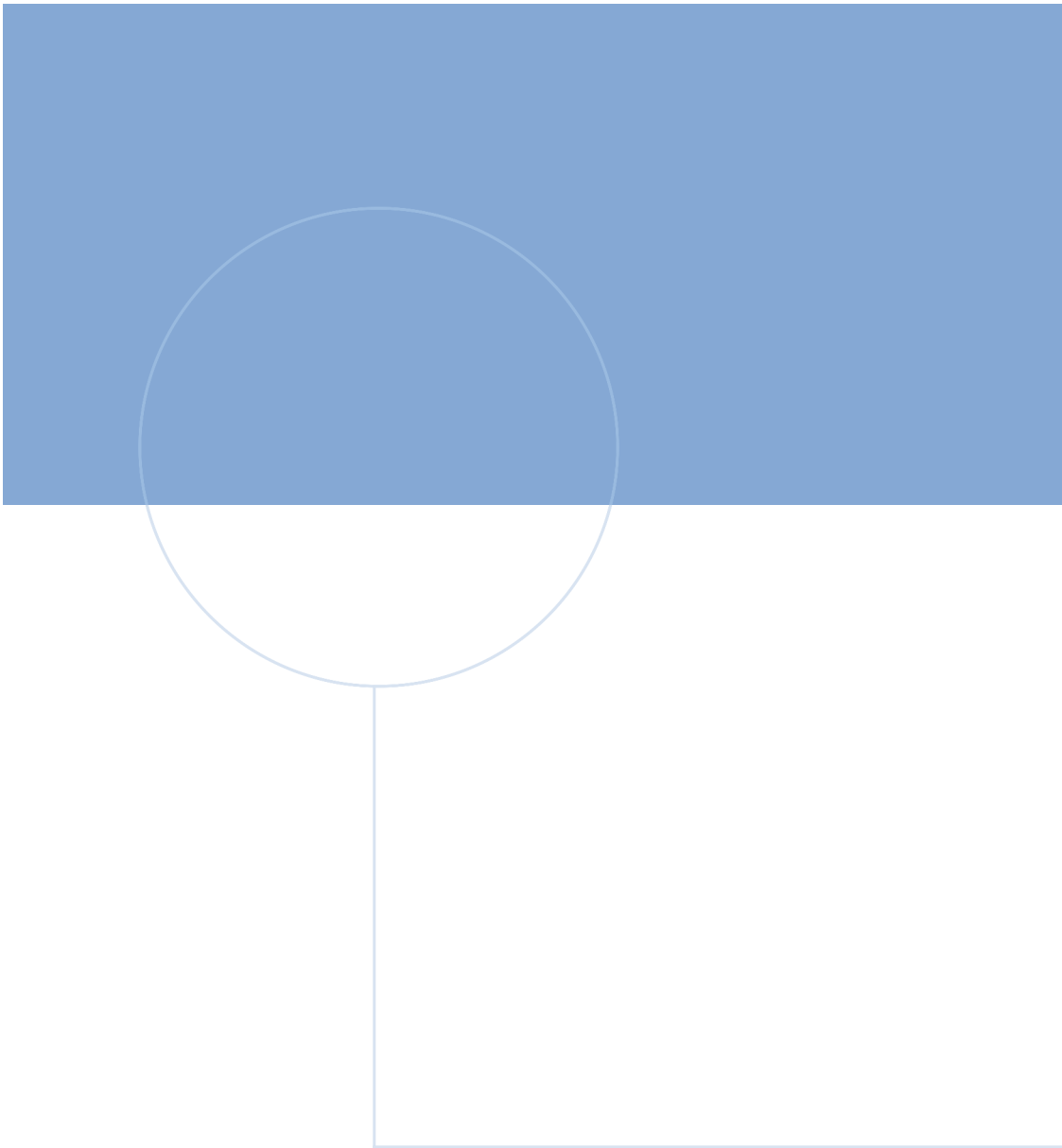
Appendix

```
Q = ∅
for trackedObj ∈ trackedObjs do
    for detectedObj ∈ detectedObjs do
        diff = BoundingBoxDifference(trackedObj, detectedObj)
        if diff ≤ differenceThreshold then
            Q.InsertWithPriority({trackedObj, detectedObj}, diff)
        end
    end
end
while Q ≠ ∅ do
    {trackedObj, detectedObj} = Q.ExtractMin()
    if not trackedObj.matched and not detectedObj.matched then
        Match(trackedObj, detectedObj)
        trackedObj.matched = True
        detectedObj.matched = True
    end
end
for detectedObj ∈ detectedObjs where not detectedObj.matched do
    StartNewTrack(detectedObj)
end
```

Algoritme 1: Algoritme for objektmatching

$dAvg = (d1 + d2)/2$
$pAvg = (p1 + p2)/2$
$rotAx = dAvg \times pAvg$
$angle = \mathbf{acos}((dAvg \cdot pAvg)/(dAvg.norm * pAvg.norm))$
$rot1 = \mathbf{RotationFromAxisAngle}(rotAx, angle)$
$d1 = d1 * rot1$
$d2 = d2 * rot1$
$n1 = d1 \times d2$
$n2 = p1 \times p2$
$angle = \mathbf{acos}((n1 \cdot n2)/(n1.norm * n2.norm))$
$axisCheck = n1 \times n2$
if $axisCheck \cdot pAvg < 0$ **then**
  $angle = angle * -1$
end
$rot2 = \mathbf{RotationFromAxisAngle}(pAvg, angle)$
$totalRotation = rot1 * rot2$

Algoritme 2: Algoritme for sammenstilling av vinkler



NTNU
Norwegian University of
Science and Technology

FFI Forsvarets
forskningsinstitutt